CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingenería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Creación y visualización de métricas ágiles mediante el uso de herramientas de mashup

Autor: Alejandro Rodríguez Fraga

Director: Rafael Fernández Gallego

MADRID, JUNIO 2016

RESUMEN

Resulta cada vez más habitual que los equipos de desarrollo software que trabajan siguiendo metodologías ágiles utilicen diversas herramientas online para sus tareas de gestión del proyecto, control del código o integración continua. Ejemplos de estas herramientas son Jira, Github o Jenkins. El problema de este escenario es que el equipo genera una gran cantidad de información de proyecto totalmente dispersa, y resulta por tanto indispensable su posterior recopilación para poder monitorizar el proceso de desarrollo y generar las métricas ágiles necesarias para su análisis.

Este trabajo propone una solución a este problema usando la herramienta de mashup Wirecloud, desarrollando una serie de componentes que permiten a los usuarios configurar de forma flexible la información que obtienen y cómo la representan, permitiendo generar distintas métricas ágiles que se ajusten a sus necesidades.

Los componentes desarrollados se dividen en distintas categorías:

- Harvesters (recolectores de datos) que se ocupan de obtener la información de las herramientas ágiles online.
- Splitters, cuya función es obtener propiedades concretas de los datos obtenidos por los harvesters.
- Componentes que realizan transformaciones sobre listas, encargados de agrupar, filtrar los datos y realizar operaciones sobre los datos
- Componentes de Representación gráfica, cuya función es transformar los datos de forma que estos representen gráficas o tablas de datos y mostrarlos al usuario.

Los componentes de Wirecloud envían los datos que generan entre sí mediante unas conexiones configurables, de forma que permite (al ser los componentes desarrollados genéricos, aceptando una gran variedad de datos de entrada) generar salidas que se ajusten a las necesidades concretas del proyecto analizado sin tener que desarrollar nuevos componentes.

ABSTRACT

It is increasingly common for software development teams that work following agile methodologies to use various online tools to help with project management, source control and continuous integration task. Some of these tools are Jira, Github or Jenkins. The main issue working like these is that the developer team generates a huge amount of data that ends up scattered over all the online tools used by the developer team, and it's important to collect it back in order to monitor the development process and generate the needed metrics to analyze the development process.

This project proposes a solution to this issue using the mashup tool Wirecloud, developing components that would allow users to setup the data that will be harvested and how the data will be showed, allowing the users to create the metrics needed for their particular project.

The Wirecloud components that will be developed are divided in four categories:

- Harvesters, whose function is to gather data from the online tools used by the agile projects.
- Splitters, whose function is to get properties out of the data obtained through the harvesters.
- Data Transformation components, whose function is to filter and transform data to get useful data out of it.
- Graphical Representation components, whose function is to transform data in order to represent graphs and tables, and plot them to the user.

Wirecloud components send their output data through configurable connections called wires, that, since the developed components are generic and accept a great variety of input data; allow the user to modify the final output by changing the connections between components, without the need to develop new components.

Índice

1 INTRODUCCION Y OBJETIVOS

Hoy en día, la mayoría de los proyectos software siguen metodologías de desarrollo ágil, como pueden ser SCRUM[2] eXtreme Programming (XP)[3]. Una de las características clave de estas metodologías es la visibilidad de la información inherente del proyecto. Toda la información suele estar abierta y disponible a todo el mundo, facilitando con ello la agilidad y promocionando la mejora continua. Compartiendo entre toda la organización los éxitos y los fracasos cosechados por el equipo de desarrollo, se intenta fomentar el crecimiento individual y la creación de mejores soluciones técnicas a los problemas.

Todos los miembros del equipo de desarrollo, en sus procesos, crean continuamente grandes cantidades de información la cual se almacena en diferentes herramientas ágiles (por ejemplo, sistemas de gestión de proyectos, de control del código fuente, de integración continua, de despliegue, o de monitorización del rendimiento de la aplicación). El código fuente, los informes de bugs, o los elementos del backlog son ejemplos de información generada.

Las metodologías ágiles ofrecen muchas oportunidades para inspeccionar el proceso de desarrollo y realizar todos los ajustes que sean necesarios para mejorar la eficiencia y la productividad del equipo. Por ejemplo, en SCRUM se realizan reuniones diarias para conocer el estado del proceso, más las reuniones de final del sprint y de retrospectiva. Esto implica, que cada día, los miembros del equipo de desarrollo SCRUM necesitan conocer cierta información clave para poder realizar sus tareas satisfactoriamente, para lo cual hacen uso de gráficas cada vez más conocidas, como pueden ser los diagramas de "burndown", de "velocidad", de objetivos del equipo, los "taskboards", etc.

El problema es que cada rol requiere distinto tipo de información: por ejemplo, un desarrollador querrá ver los trabajos que quedan por realizar (issues), mientras que el "scrum master" querrá ver el progreso del sprint o la carga de trabajo de los desarrolladores, y además, esta información está cada vez más dispersa en las diferentes herramientas ágiles online utilizadas (por ejemplo, JIRA[13], Jenkins[12], GitHub[10], etc.), por lo que el proceso de obtener y componer la información para generar métricas ágiles que después poder monitorizar y evaluar resulta cada vez más complicado.

Es por esto que el presenta Trabajo Fin de Grado se plantea, a través de los objetivos presentados en el apartado siguiente, hacer uso de la herramienta de mashup Wirecloud[8], para facilitar la creación de distintos cuadros de mando o dashboards de monitorización que permitan fácilmente a los distintos usuarios configurar y visualizar la información que necesitan para su rol, de forma que puedan acceder cómodamente a los datos de utilidad y visualizar de forma sencilla las métricas que desean.

1.1 Contexto del proyecto

En esta sección se describen los elementos principales del contexto del proyecto:

- Laboratorio CoNWeT, laboratorio en el que se desarrolla este proyecto y desarrolladores de la herramienta de mashup Wirecloud que se usará.
- FIWARE, proyecto europeo cuyo análisis es el caso de uso de este proyecto
- Las metodologías ágiles, que son el enfoque principal del proyecto al girar en torno al análisis de proyectos que se guían por estas metodologías de desarrollo.

1.1.1 FIWARE

FIWARE[6] es un proyecto europeo que tiene como objetivo crear la plataforma que sirva como núcleo de la Internet del Futuro, facilitando el desarrollo eficiente de aplicaciones y servicios proveyendo una arquitectura abierta y pública que permita satisfacer las necesidades de los desarrolladores.

El proyecto FIWARE está compuesto por múltiples grupos de desarrolladores, llamados capítulos, que no trabaja en un mismo lugar físico y que requieren coordinarse entre sí para realizar de forma efectiva el desarrollo del proyecto. Debido a esto, la coordinación puede ser difícil y se usan herramientas ágiles para facilitarla. En el caso de FIWARE la herramienta de coordinación es Jira, en la que se crean issues que identifican los objetivos globales y los objetivos de cada área del proyecto, permitiendo así al equipo de desarrolladores conocer el estado del general del proyector y coordinarse entre sí.

Figure 1: Logo de FIWARE

1.1.2 Laboratorio CoNWeT

El laboratio CoNWeT[7] (Computer Networks and Web Technologies Laboratory) es un laboratorio que pertenece al grupo de investigacion CETTICO de la de Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid. El laboratorio CoNWeT se centra en la investigación sobre la Internet

del Futuro y sobre tecnologías web. En concreto, el laboratorio CoNWeT ha liderado el desarrollo de los capitulos de FIWARE de Apps y Data Delivery, así como el desarrollo de la herramienta de mashup Wirecloud.

El grupo de investigación CETTICO ha sido valorado en la decimotercera posición dentro de los 169 grupos de investigación reconocidos por la UPM.

Figure 2: Logo del CoNWeT

1.1.3 Metodologías ágiles

Las metodologías ágiles son un conjunto de técnicas para la gestión de proyectos surgidas en el ámbito del desarrollo software, aunque también son aplicadas en otros ámbitos.

Las metodologías ágiles están basadas en el desarrollo iterativo e incremental, realizando así ciclos continuos de desarrollo, en los que los requisitos y funciones a desarrollar van evolucionando con el tiempo. Uno de los principales énfasis de las metodologías ágiles recae en la comunicación directa entre los miembros del equipo, teniendo reuniones diarias por ejemplo en Scrum.

Cada uno de estos ciclos de desarrollo es llamado sprint. Al comienzo de cada sprint se deciden las funciones del proyecto a desarrollar, siendo estas funciones representadas en la forma de los issues del proyecto.

Las metodologías ágiles tienen un gran enfoque a la comunicación entre los miembros, en el caso de Scrum teniendo reuniones diarias para valorar el progreso del sprint y analizar posibles dificultades que puedan surgir.

1.2 Motivación del proyecto

Para la visualización del progreso en el caso de uso planteado (El seguimiento del progreso del desarrollo ágil del Proyecto FIWARE), se utilizaba una página web3 en la que obtenían datos de la herramienta ágil Jira para posteriormente generar gráficas que permiten visualizar el progreso del proyecto y del sprint actual. Este método tiene una serie de desventajas , como la limitada capacidad de la aplicación, al obtener datos únicamente de Jira y no del resto de herramientas ágiles disponibles como Github, y la imposibilidad de configurarlo, no siendo posible por tanto obtener

información sobre otros proyectos ni información sobre sprints anteriores del desarrollo del proyecto. Otra gran desventaja que tenía la aplicación eran unos grandes tiempos de carga requeridos cada vez que se entraba en la página y el servicio obtenía los datos de Jira y calculaba los gráficos a mostrar.

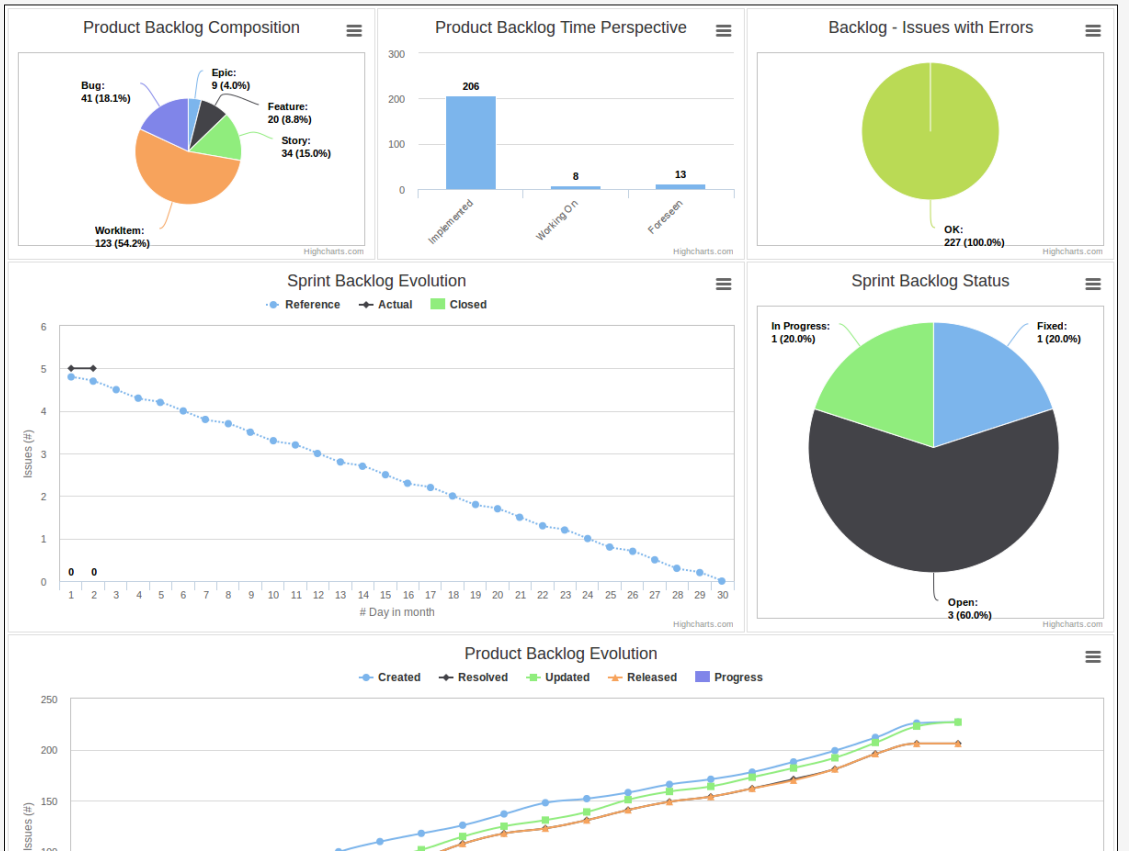


Figure 3: Dashboards usado por Fiware para el seguimiento del proyecto

1.3 Objetivos

Este proyecto plantea crear todos los componentes necesarios para poder representar la misma información que la que se encuentra en dicha aplicación, pero añadiendo la posibilidad de usar otras herramientas ágiles, concretamente Github, Gitlab, Jira y Jenkins, y permitiendo al usuario generar nuevas gráficas que se ajusten a sus necesidades al poder recomponer las conexiones entre los distintos componentes, permitiendo así crear nuevas gráficas y tablas que permitan mostrar detalles concretos del desarrollo del proyecto que se ajusten a los deseos del usuario.

Al realizar esto con la herramienta de mashup Wirecloud, se permite que haya una gran libertad en el uso de los componentes permitiendo obtener y modificar los tipos de datos mostrados para otorgar flexibilidad y permitir a la aplicación ajustarse a las necesidades concretas del usuario.

El proyecto plantea el diseño e implementación de los siguientes tipos de componentes:

- Harvesters (recolectores de datos) que se ocupan de obtener la información de las herramientas ágiles online, recolectando la información que usarán el resto de componentes.

- Splitters, cuya función es obtener propiedades concretas de los datos obtenidos por los harvesters, permitiendo así seleccionar la información deseada de entre toda la información disponible.

- Componentes que realizan transformaciones sobre listas, encargados de agrupar, filtrar los datos y realizar operaciones sobre los datos, permitiendo así calcular los datos deseados que posteriormente serán representados por los componentes de representación gráfica.

- Componentes de Representación gráfica, cuya función es transformar los datos de forma que estos representen gráficas o tablas de datos y mostrarlos al usuario.

Uno de los principales criterios tenidos en cuenta durante el desarrollo de los componentes del proyecto ha sido el hacerlos lo más genéricos posible, de forma que cada componente sean compatibles con todos los demás componentes, para así lograr permitir un mayor número de posibilidades a la hora de interconectar los componentes y crear mashups que se puedan ajustar de forma flexible a las necesidades del usuario final, intentando a su vez mantener una gran sencillez en los componentes, de forma que no se requieran conocimientos avanzados a la hora de usar estos componentes, ya que importantes roles del proyecto, como el Product Owner, no tienen por que tener ningún conocimiento de programación ni estar acostumbrado al uso de herramientas de mashup como Wirecloud.

2 ESTADO DEL ARTE

En este apartado se va a describir las tecnologías que se han utilizado en el desarrollo de este proyecto y las herramientas ágiles más usadas y que explotarán los componentes de Wirecloud desarrollados.

2.1 Scrum

Scrum[2] es una metodología de desarrollo ágil de software basada en el desarrollo iterativo e incremental.

Scrum está basado en torno a la idea de que durante el desarrollo pueden surgir problemas imprevistos que impidan el progreso en el desarrollo de una funcionalidad y en la idea de que el cliente puede querer cambiar los requisitos del producto final en cualquier momento.

Para evitar los conflictos anteriores, Scrum intenta ser flexible para poder adaptarse a todos estos imprevistos, centrándose en la entrega rápida y continua de resultados parciales (Entregando funcionalidades del producto final en vez de esperar a tener el producto entero acabado).

En Scrum, el desarrollo del producto esta dividido en sprints, cuyo objetivo es acabar y entregar al cliente las funcionalidades elegidas a desarrollar durante el sprint (a estas funcionalidades a desarrollar durante un sprint se las conoce como "Sprint Backlog", mientras que el "Product Backlog" sería la totalidad de funcionalidades que por desarrollar de las que constara el producto final). Antes de cada sprint tiene lugar una reunión de planificación en la que se eligen los elementos que serán desarrollados durante el sprint, y, al final del sprint, se realiza otra reunión para valorar el trabajo realizado durante el sprint. Normalmente estos sprints tienen una duración de una a cuatro semanas.

Durante cada sprint también se realizan cortas reuniones diarias cuya función es descubrir y decidir cómo solucionar aquellos problemas que puedan impedir el progreso del desarrollo.

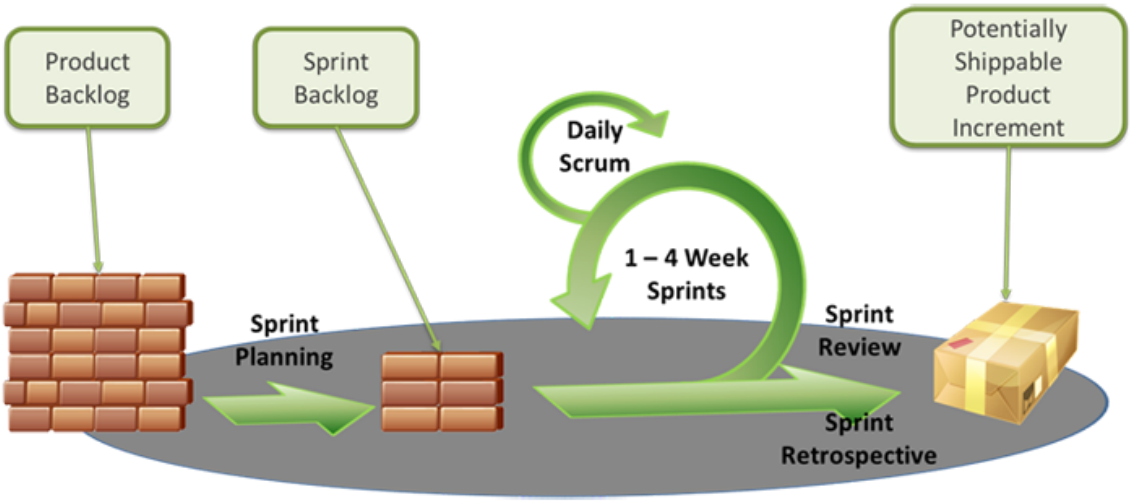


Figure 4: Estructura del ciclo de desarrollo de SCRUM

En Scrum existen tres roles principales que cumplen los miembros implicados en el proyecto:

- Product Owner: Es el encargado de vender el producto, siendo el que se ocupa de interactuar con el cliente para descubrir nuevos requisitos y mostrar el progreso del proyecto.
- Scrum master: Es el encargado de asegurar el progreso de proyecto, ayudando a los desarrolladores a seguir la metodología de Scrum y determinar cómo eliminar los impedimentos que surjan durante el desarrollo.
- Desarrollador: Son los encargados de desarrollar el producto, siendo responsables de que al final de cada sprint haya valor ganado.

Figure 5: Logo de Scrum

2.2 Tecnologías usadas

En esta sección son descritas el conjunto de tecnologías usadas en el desarrollo del proyecto.

2.2.1 Wirecloud

Wirecloud[8] es una plataforma de mashup, pensada para permitir a usuarios finales sin conocimientos de programación crear fácilmente aplicaciones desde las que poder ver información de interés de múltiples fuentes (mashup). Dentro de Wirecloud existen dos tipos de componentes:

- Widgets: Estos componentes son los encargados de la representación visual de la información, por tanto, en la vista de la aplicación, estos serán los únicos que se tendrán representación gráfica en la vista del usuario.
- Operadores: Estos componentes son los encargados de modificar y obtener los datos de diversas fuentes, como por ejemplo de algún servicio web que provea información de interés para el usuario final.

En Wirecloud hay dos vistas de la aplicación, la primera siendo el "dashboard", que sería la vista final de la aplicación, en la que el usuario visualiza la información e interactúa con la interfaz gráfica de los widgets, y la vista de "wiring", en la que el usuario puede interconectar los widgets y operadores entre sí, definiendo de esta forma el flujo que la información seguirá dentro de la aplicación creada, permitiendo conectar de múltiples formas los componentes y así crear fácilmente diferentes aplicaciones que se ajusten a las necesidades del usuario.

Figure 6: Logo de Wirecloud

2.2.2 HTML

HTML (HYperText Markup Language)[16] es un lenguaje de marcado usado para crear páginas web. Este lenguaje es el estándar para la visualización de páginas web y es el que ha sido adoptado por todos los navegadores actuales.

Un documento HTML está formado por una serie de etiquetas que representan los atributos, y el contenido, que se encuentra entre la apertura y el cierre de la etiqueta.

La interacción entre HTML y otros lenguajes de programación se realiza mediante el DOM (Document Object Model), que realiza la función de interfaz entre HTML y el código usado en el navegador para interactuar con la página (normalmente JavaScript)

Figure 7: Logo de HTML

2.2.3 Javascript

JavaScript[15] es un lenguaje de programación de alto nivel e interpretado que normalmente es usado por los navegadores para crear páginas web dinámicas. JavaScript es un lenguaje de programación multiparadigma, ya que soporta programación orientada a objetos, programación imperativa y programación funcional. Aunque normalmente JavaScript es usado en los navegadores para ejecutar código en el cliente de la página web y así crear páginas dinámicas, también se utiliza en otros ámbitos, un ejemplo de esto es Node.js.

JavaScript es el principal lenguaje en el que se programan los componentes para Wirecloud, por lo que será el lenguaje más utilizado durante el desarrollo del proyecto.

Figure 8: Logo de Javascript

2.2.4 JQuery

JQuery[14] es una librería de JavaScript basada en código abierto y libre. JQuery la librería de JavaScript más utilizada, y que ofrece una interfaz que permite in-

teractuar fácilmente con los documentos HTML, manipular el árbol DOM de un documento HTML.

Figure 9: Logo de Jquery

2.2.5 CSS

CSS (Cascading Style Sheets) es un lenguaje usado principalmente para describir la representación visual del HTML de una página web, aunque puede usarse para describir cualquier tipo de lenguaje de marcado, como por ejemplo XML.

CSS permite separar el contenido del documento con su representación visual, permitiendo así flexibilidad en cuanto a su representación, pudiendo usar distintos estilos para la misma información dependiendo del contexto.

Figure 10: Logo de CSS

2.2.6 Git

Git[5] es un sistema distribuido de control de versiones, que permite la cómoda gestión del código.

Git almacena una lista de los cambios aplicados a los ficheros del repositorio, de forma que puede recuperar cualquier versión de los mismos a partir de versiones anteriores y la cadena de cambios aplicados a dichas versiones desde su creación.

Git permite que los clientes tengan en local una copia del repositorio sobre el que trabajan, para luego poder subir los cambios realizados al mismo mediante un "commit", de forma que se pueda trabajar sin conexión y publicar los cambios una vez terminados.

Figure 11: Logo de Git

2.2.7 Highcharts

Highcharts[4] es una librería para representar gráficas usando JavaScript que ofrece sus servicios de forma gratuita para uso no comercial. Todas las gráficas que se crearán en este proyecto usarán esta librería.

Figure 12: Logo de Highcharts

2.3 Herramientas ágiles

Estas son las principales herramientas ágiles usadas, aunque hay muchas más, como por ejemplo bitbucket, que es una herramienta similar a Github, las definidas en esta sección serán las únicas de las que se obtendrán datos con los que analizar los proyectos que las usen, ya que son las más usadas y las que son utilizadas en el caso de uso planteado.

2.3.1 Github

Github[10] es un servicio web que permite alojar en el repositorios de Git, de forma que se pueda almacenar en él los elementos del proyecto sobre los que se va a hacer control de versiones.

Github ofrece repositorios públicos de forma gratuita, por lo que es una herramienta muy utilizada en proyectos open-source. También dispone de repositorios privados, pero estos son de pago.

Además de la funcionalidad de Git, ofrece un sistema de "issues" (Unidad de trabajo para realizar una mejora en un Sistema Informático [link]) y "milestones"

(puntos específicos en el desarrollo de un proyecto [link]), permitiendo realizar un seguimiento del proyecto y establecer objetivos e hitos a realizar.

Figure 13: Logo de Github

2.3.2 Gitlab

Gitlab[11] es un software opensource que permite montar servicios para alojar repositorios de git, similar a Github. Sin embargo, al poder montar el servicio en un servidor propio, permite que los repositorios creados sean privados, por lo que Gitlab es una de las herramientas de gestión de versiones más usada por empresas privadas.

Al igual que Github, dispone de un sistema de issues y milestones.

Figure 14: Logo de Gitlab

2.3.3 Jira

Jira[13] es una aplicación para el seguimiento de errores y gestión de proyectos mediante el uso de issues. En cuanto a la gestión de issues es mucho más complejo que Gitlab y Github, aunque este no posee ningún tipo de control de versiones sobre el código.

En Jira se pueden realizar funciones complejas sobre los issues cómo asignarles un flujo para controlar el estado de la issue, obligando a que las issues tengan que transicionar entre los estados tal y como esté definido en dicho flujo, permitiendo de esta forma establecer mecanismos de control de calidad sobre las issues y el código asociado a las mismas.

Figure 15: Logo de Jira

2.3.4 Jenkins

Jenkins[12] es un software open source de integración continua basado en la ejecución de tests de calidad software para evaluar una "build" del proyecto. Jenkins permite automatizar la ejecución de dichos test de calidad, de forma que se pueda ver rápidamente si los cambios realizados sobre el código son correctos o no, la velocidad de ejecución de las pruebas, y obtener información sobre la tendencia de los tests, para ver si se está empeorando o mejorando la calidad del software desarrollado.

Jenkins dispone de varios tipos de trigger (gatillos) que permiten elegir en momentos se ejecutan los tets sobre el código, como por ejemplo al realizar un commit a un repositorio Git, permitiendo su uso automáticoen combinación con otras herramientas online como Github.

Figure 16: Logo de Jenkins

3 DESARROLLO

El desarrollo del proyecto ha constado de un primer análisis de las gráficas ágiles más usadas a su vez que aquellas usadas en la solución que se utilizaba previamente en el caso de uso planteado.

Tras ello, se desarrollaron los componentes de la plataforma Wirecloud necesarios para poder obtener, transformar y mostrar toda esta información de una forma sencilla y genérica, de forma que sea fácil modificar las configuraciones y conexiones entre los componentes desarrollados para así representar otros tipos de información que puedan satisfacer las necesidades concretas de cualquier proyecto.

3.1 Análisis de las gráficas para evaluar el progreso de proyectos ágiles

Las gráficas son una herramienta muy útil para analizar el progreso de los proyectos ágiles ya que permiten rápida y cómodamente ver y valorar distintos aspectos del proyecto analizado.

Los tres tipos de gráficas que serán utilizados para el desarrollo de este proyecto serán los gráficos de columnas, de tarta y de líneas, ya que estas gráficas, al ser simples y fáciles de entender y analizar permiten representar una gran variedad de información con operadores genéricos que sepan generar este tipo de gráficas.

Las principales gráficas específicas usadas para analizar proyectos ágiles son las gráficas de burndown, reliability, workload y project backlog.

3.1.1 Gráfica burndown

La gráfica de burndown de un proyecto ágil permite observar el progreso de la carga de trabajo de un proyecto durante un sprint concreto, mostrando cada día del sprint la cantidad de issues asignadas al sprint que quedan por cerrar.

La gráfica Burndown es una gráfica de líneas formado por dos series de datos, una representando el progreso real realizado durante el sprint analizado y otra el progreso estimado para el mismo sprint, siendo la primera una representación de las issues que quedan por cerrar en cada día del sprint, es decir, la carga de trabajo restante, y la segunda es una serie constante que marca el progreso diario medio necesario para completar todas las issues del sprint a tiempo. Se puede observar un ejemplo de este tipo de gráficas en la figura 17

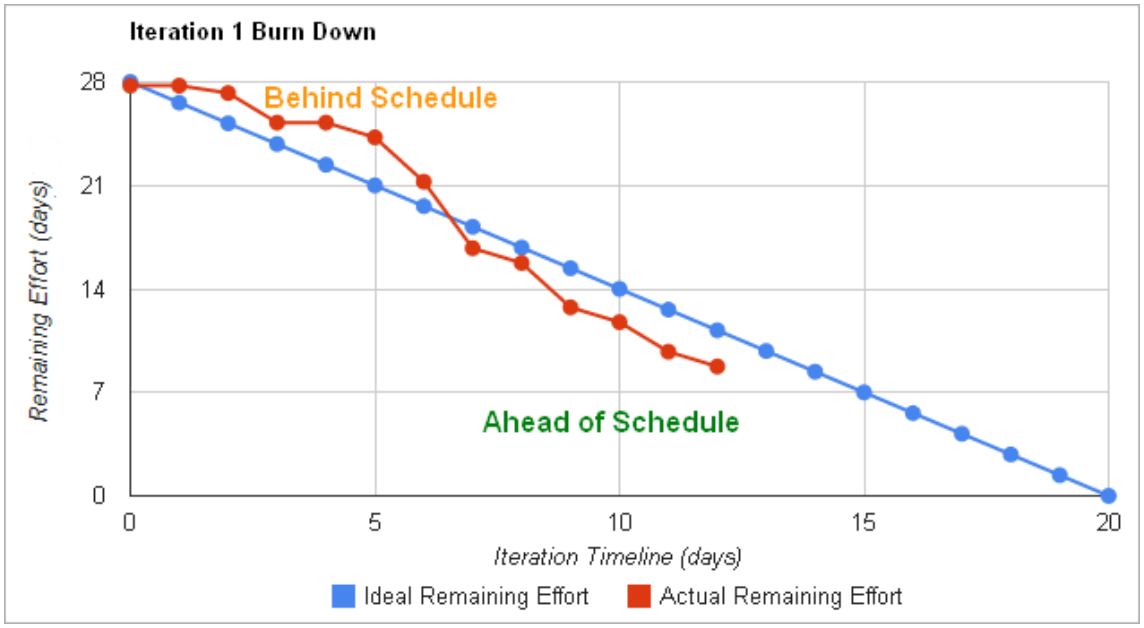


Figure 17: Ejemplo de gráfica de burndown

3.1.2 Gráfica de reliability

La gráfica de reliability (gráfica de fiabilidad) sirve para analizar la relación entre issues abiertas y cerradas de los miembros del proyecto, permitiendo, por ejemplo, analizar qué miembros del proyecto no lograron cerrar todas las issues que tenían asignadas al finalizar el sprint, para permitir repartir mejor la carga de trabajo para la siguiente iteración del desarrollo ágil.

Esta métrica se suele representar mediante una gráfica de barras, estando cada barra asociada a un miembro del proyecto. El eje Y de la gráfica representa el porcentaje de issues, por tanto, al estar cada una de las barras divida en una primera parte de issues completadas, y una segunda parte de issues no completadas, se puede apreciar facilmente la cantidad de issues no completadas en relación a las completadas. Un ejemplo de esta gráfica es la figura 18

Figure 18: Ejemplo de gráfica de reliability

3.1.3 Gráfica de workload

La gráfica de workload (gráfica de carga de trabajo) permite observar la carga de trabajo asociada los miembros del proyecto, permitiendo a partir de ella saber fácilmente qué miembros del proyecto tienen más carga de trabajo que otros para poder repartir la carga y así aumentar el rendimiento del grupo.

Esta gráfica se representa normalmente en forma de gráfica de tartas, en la que cada sección se corresponde a uno de los miembros del proyecto, y dependiendo el tamaño de la sección del número de issues asignadas. Un ejemplo de esto es la figura 19. Esta gráfica también se puede representar por columnas, de forma que se pueda

ver el workload de los miembros del proyecto a lo largo de los meses, habiendo por tanto una columna por cada miembro del proyecto por cada mes, y siendo el valor de esta el número de issues asignadas durante dicho mes.

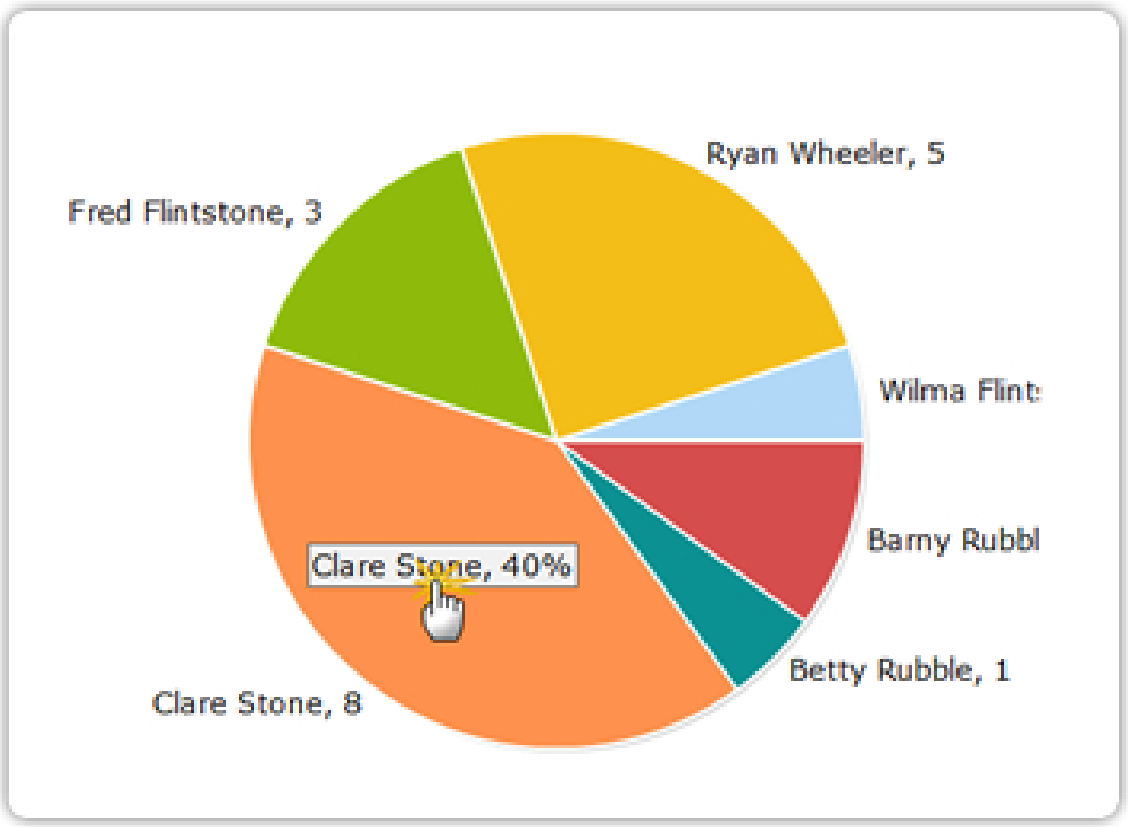


Figure 19: Ejemplo de gráfica de Workload

3.1.4 Gráfica de Project Backlog Evolution

Esta gráfica permite analizar el crecimiento del proyecto sobre el tiempo, así como su progreso. De esta forma se puede ver fácilmente por ejemplo si el proyecto se está quedando atrasado, teniendo cada vez más issues abiertas, o si por ejemplo se está quedando parado, al no crearse nuevas issues pese a que las pocas que se creen si se están cerrando. Esto permite evaluar el progreso del proyecto a lo largo de los meses.

Esta gráfica esta formada por series de datos representados en forma de líneas, siendo estos las issues creadas, cerradas, actualizadas y liberadas cada mes, y una serie de datos representada en forma de columnas que indica el número de issues cerradas ese mes.

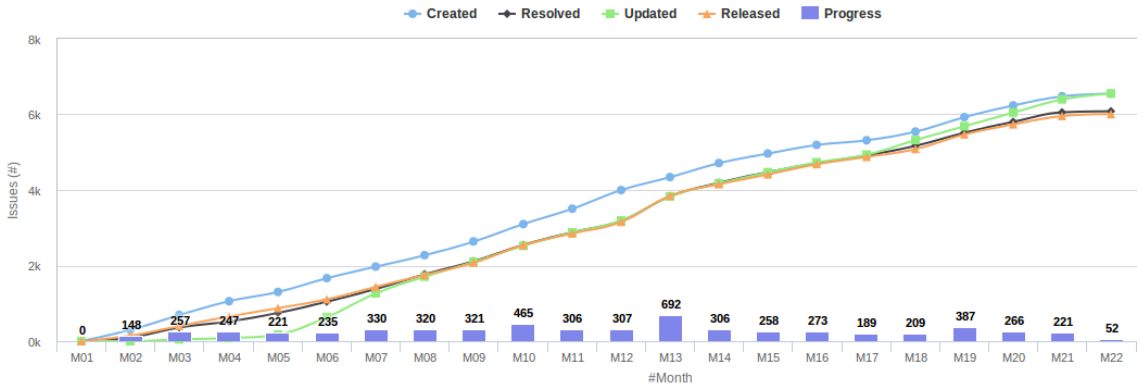


Figure 20: Ejemplo de gráfica de Project Backlog Evolution

3.2 Componentes desarrollados

En esta sección son descritos los componentes que se han desarrollado para lograr satisfacer los objetivos del proyecto.

Los componentes han sido desarrollados en el orden que seguirán en los mashups desarrollados con ellos, siendo por tanto desarrollados primero los harvesters, seguidos de los operadores usados para transformar los datos y por último los componentes necesarios para generar los datos de las gráficas y tablas y representarlas. De esta forma, se han desarrollado primero los componentes que no dependen de otros primero, de forma que a la hora de desarrollar un componente, este no dependa de otro componente que aún no haya sido diseñado.

3.2.1 Harvesters de datos

Los harvesters de datos (recolectores de datos) son los operadores encargados de obtener datos de las herramientas ágiles online, por tanto, son el componente principal de los dashboards ágiles desarrollados y habrá como mínimo uno siempre presente en cualquier dashboard ágil ya que son necesarios para obtener los datos a analizar.

3.2.1.1 Operador Harvester de Github

El harvester de Github es el harvester encargado de obtener datos de Github. Este harvester obtiene información sobre las issues de un repositorio, que luego es normalizada para tener el mismo formato de issue que el resto de harvesters de issues, e información sobre los commits de git. Debido a los datos ofrecidos por la API de Github, este harvester es el más completo, al permitir obtener un historial

de cambios de estado de una issue, lo que permite saber por ejemplo si fue cambiada de milestone (la cual es usada como sprint normalmente).

Este operador tiene dos outputs: uno para las issues y otro para los commits obtenidos. En cuanto a la configuración, este operador permite seleccionar el repositorio del que obtener datos, un token de autenticación OAuth2 y un usuario y contraseña en caso de no querer usar OAuth para autenticarse. También se puede elegir la cantidad de elementos a obtener.

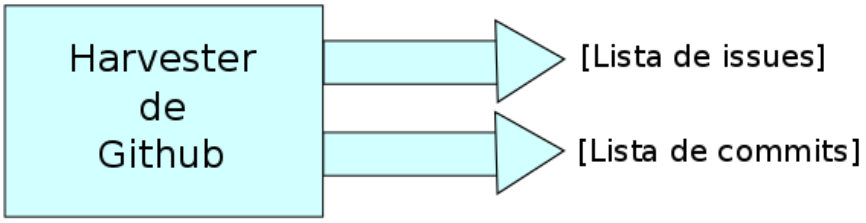


Figure 21: Flujo de datos del operador Harvester de Github

3.2.1.2 Operador Harvester de Gitlab

El harvester de Gitlab es el harvester encargado de obtener datos de Gitlab. Al igual que el harvester de Gitlab, el harvester de Gitlab obtiene información de las issues de un repositorio, que son normalizadas para tener el mismo formato que las issues de Github y las issues de Jira, e información sobre los commits de un repositorio, que también son normalizados para tener el mismo formato que los obtenidos por el harvester de Github.

Este operador tiene dos outputs: uno para las issues y otro para los commits obtenidos. En cuanto a la configuración, este operador permite seleccionar el repositorio del que obtener datos, un token de autenticación OAuth2 y un usuario y contraseña en caso de no querer usar OAuth para autenticarse. También se puede elegir la cantidad de elementos a obtener.

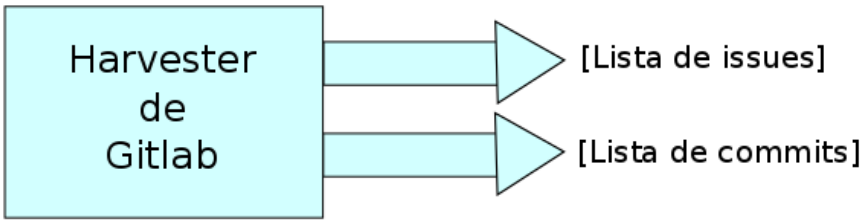


Figure 22: Flujo de datos del operador Harvester de Gitlab

3.2.1.3 Operador Harvester de Jira

El harvester de Jira se ocupa de obtener datos de la herramienta Jira, por tanto, este harvester solo obtiene issues, que normaliza como en el caso de los harvesters anteriores.

Este operador solo tiene un output, por el que se envían las issues obtenidas. Se puede configurar el servidor de Jira del que se obtendrán los datos, así como el proyecto y componentes concretos cuyos datos obtener. En cuanto a autenticación, se necesitará usar usuario y contraseña.

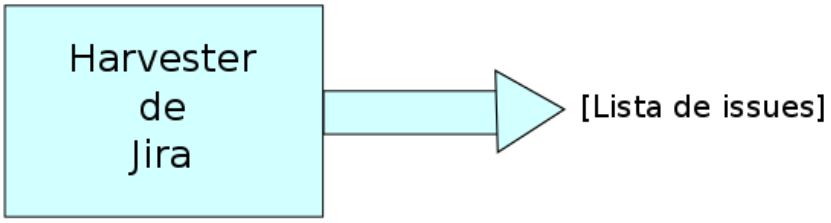


Figure 23: Flujo de datos del operador Harvester de Jira

3.2.1.4 Operador Harvester de Jenkins

El harvester de Jenkins obtiene datos de Jenkins, es decir, datos sobre las builds realizadas en él, por tanto, la salida obtenida por este operador no es compatible con la obtenida por los harvesters previos, por lo que solo se podrán usar con el

los componentes genéricos como el operador Pie Chart Generator y los que estan especialmente diseñados para su uso con este, como el operador Detailed Jenkins Test Report.

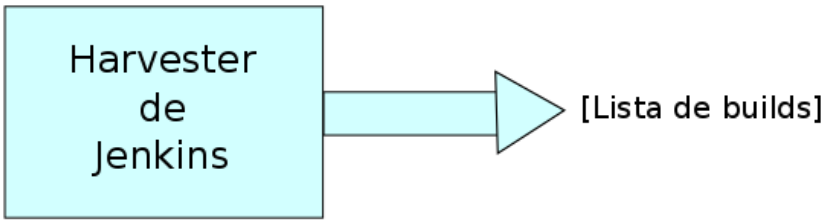


Figure 24: Flujo de datos del operador Harvester de Jenkins

3.2.1.5 Operador Detailed Jenkins Test Report

La función del operador Detailed Jenkins Test Report es obtener información más detallada sobre el conjunto de builds de Jenkins recibidas como entrada, por tanto, siempre será usado conectando su entrada con la salida del Harvester de Jenkins, ya sea esta filtrada previamente o no.

La información adicional obtenida por este operador permite obtener información sobre cada uno de los tests pasados por una build, permitiendo así comparar duraciones de las builds usando el operador Jenkins Test Time Difference.

Esta información adicional no es obtenida por el Harvester de Jenkins ya que debido a la gran cantidad de información que obtiene, aumentaría mucho el tiempo que tarda en obtener los datos, haciendo a todos los demás componentes del mashup esperar a que acabe de obtenerla, sobre todo en el caso de que se realizase sobre todas las builds obtenidas, por esto, se optó por separar esta funcionalidad en un operador nuevo y a su vez añadir un límite de builds a analizar configurable a este operador, de forma que no se sobrecargue el mashup obteniendo esta información adicional de todas las builds disponibles.

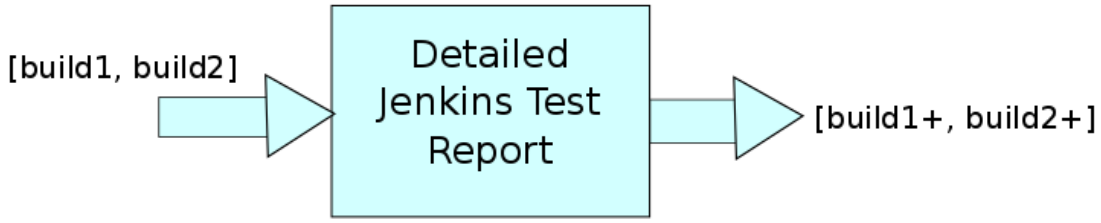


Figure 25: Flujo de datos del operador Detailed Jenkins Test Report

3.2.1.6 Operador Get Issue Closing Commit

Este operador es el encargado de obtener, a partir de una issue que esté cerrada, el commit que la cerró. Por las limitaciones ofrecidas por las APIs de las herramientas ágiles estudiadas, este operador solo funciona para los issues de Github obtenidos por el harvester de Github.

Como entrada este operador espera recibir un issue a partir del cual obtendrá el commit que lo cerró, en caso de que la issue esté cerrada y fuese cerrada por un commit. Al igual que el harvester de Github, este operador requiere configurar una autenticación para poder obtener los datos del servicio de Github. Sin embargo, en este caso no será necesario identificar el repositorio objetivo ya que esta información la obtendrá de la issue recibida como entrada.

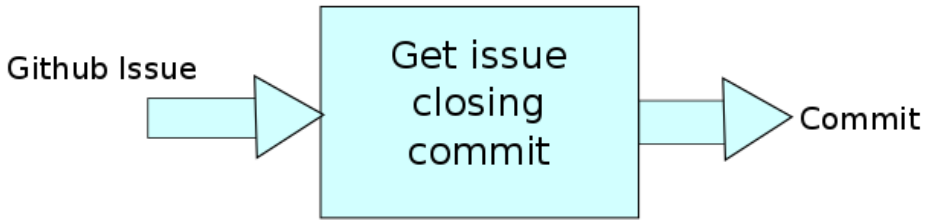


Figure 26: Flujo de datos del operador Get Issue Closing Commit

3.2.1.7 Operador GitBlame

El operador GitBlame obtiene, a partir de un commit, quien era el autor anterior de las líneas que han sido modificadas por dicho commit. Esto puede ser muy útil en un proyecto ágil para poder contactar con los autores de los errores e informarles de ellos, de forma que esto no se repita y así mejorar la calidad final del producto desarrollado.

Este operador solo funciona para Github, pese a que su API no proporcione dicha información, obteniendo los datos deseados a partir de la funcionalidad de la interfaz web de Github que permite ver el blame de un fichero. Debido a esto, solo funciona en repositorios públicos y no requiere ningún tipo de configuración, obteniendo el repositorio al que pertenece el commit a partir del propio commit recibido como entrada.

La salida generada por este operador es la que espera recibir el widget Data Viewer, siendo por tanto siempre conectada a una instancia de este. Debido a esto este operador también podría enmarcarse dentro de la categoría de operadores descritos en la sección 3.2.5.2 de este documento.

La tabla resultante consta de las siguientes columnas: autor de los cambios sobrescritos, fichero del cambio y línea del fichero modificada. Cada fila de la tabla se corresponderá con uno de los cambios realizados por el commit.

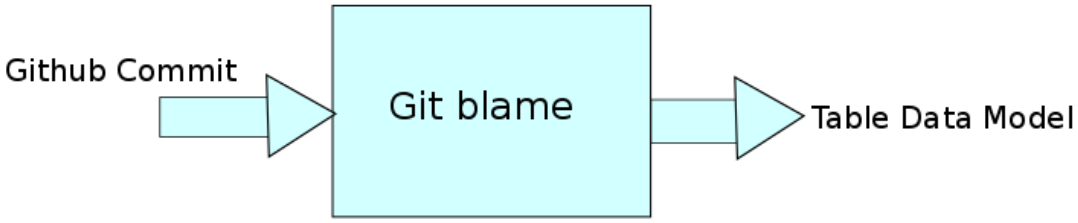


Figure 27: Flujo de datos del operador GitBlame

3.2.2 Filtrado de datos

Para filtrar los datos obtenidos se utilizan dos operadores que normalmente serán usados en conjunto, uno para elegir las condiciones por las que filtrar, y otro para realizar el filtrado de los datos que recibe.

3.2.2.1 Widget Set Generic Filter Conditions

Este widget es el encargado de elegir los filtros a aplicar a los datos. Al ser un widget, tiene representación dentro de la vista del usuario, que tendrá que usar la interfaz ofrecida por este widget para seleccionar los filtros a usar mediante una serie de desplegables, cada uno de ellos siendo un filtro a aplicar, en los que se puede elegir las condiciones para el filtro que representan.

Este componente es genérico ya que obtiene los distintos filtros disponibles para la entrada obtenida a partir de metadatos almacenados en la misma. Estos metadatos son añadidos a los datos por los harvesters que generaron dichos datos. A partir de estos metadatos obtiene los distintos tipos de filtros, mientras que a partir de los datos obtiene las distintas condiciones disponibles dentro de los datos para los tipos de filtros disponibles.

Los filtros seleccionados son persistentes para un mismo tipo de información, de forma que este componente se puede usar para filtrar la información de forma estática, configurándolo la primera vez y no volviendo a modificarlo, o de forma dinámica, cambiando el filtro para enviar distintos tipos de información al resto de componentes del dashboard.

Este widget tiene por tanto una entrada: la información de la que obtener los filtros disponibles, y una salida: las condiciones de filtrado que requiere el operador and-filter para funcionar.

3.2.2.2 Operador And Filter

Este operador es el encargado de filtrar la información en base a una lista de condiciones de filtrado que obtiene principalmente del anterior widget. En base a estos filtros genera una salida que contiene únicamente los elementos de la entrada que cumplen todos los filtros recibidos.

Tiene dos entradas, una para las condiciones de filtrado y otra para los datos a filtrar, y una única salida por la que envía los datos filtrados.

3.2.3 Operaciones sobre listas de datos

Estos operadores realizan cálculos básicos sobre listas, transformándolas para así obtener los elementos deseados para los demás componentes del mashup desarrollado.

3.2.3.1 Operador Packlist

Este operador genera como salida una lista en la que junta los elementos recibidos como entrada.

Tiene dos entradas, por las que puede recibir elementos u otras listas. La salida generada dependerá de la entrada recibida, creando una nueva lista de listas o

elementos en el caso de que ambas entradas sean del mismo tipo (dos listas o dos elementos), o añadiendo valores a la lista recibida en el caso de que solo una de las entradas sea una lista.

Este operador se podrá usar encadenado con otro operador Packlist, en cuyo caso siempre añadirá a la lista recibida del anterior Packlist los elementos recibidos por la otra entrada.

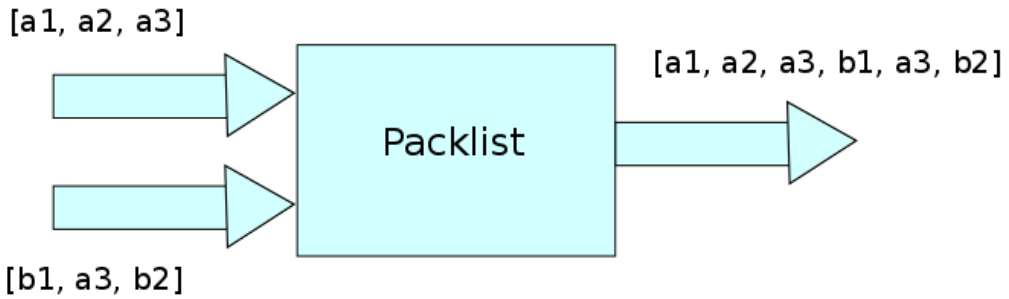


Figure 28: Flujo de datos del operador Packlist

3.2.3.2 Operador List Union

Operador encargado de calcular la unión de conjuntos entre dos listas (conjuntos de elementos), recibiendo como entrada dos listas de elementos y enviando por su salida una lista que contiene todos los elementos únicos de las dos listas recibidas.

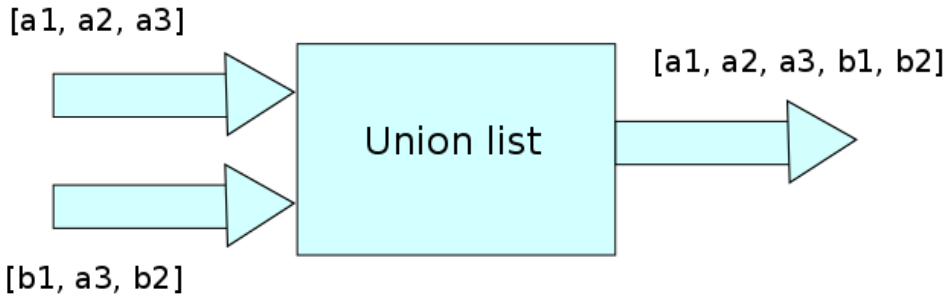


Figure 29: Flujo de datos del operador List Union

3.2.3.3 Operador List Intersection

Este operador calcula la intersección de conjuntos entre dos listas, generando en la salida una lista de elementos que contiene únicamente aquellos elementos que aparecen en ambas listas.

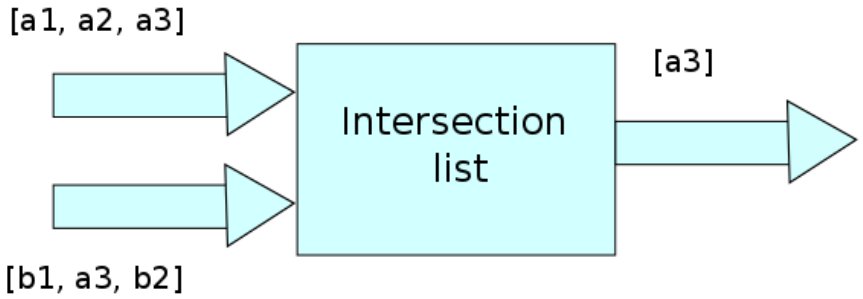


Figure 30: Flujo de datos del operador List Intersection

3.2.3.4 Operador List Difference

Este operador realiza la diferencia de conjuntos entre dos listas, enviando por su salida una lista con aquellos elementos que solo aparecen en una de las dos listas recibidas como entrada.

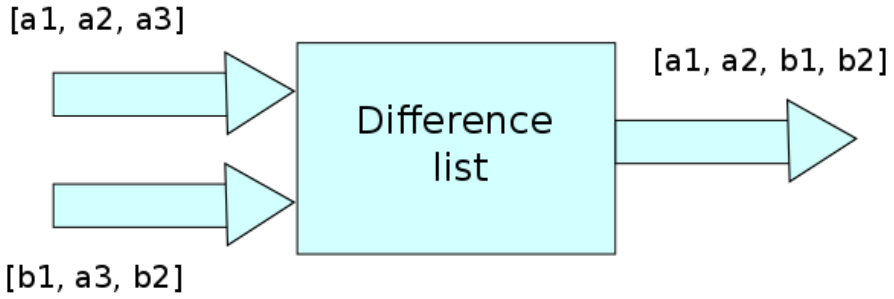


Figure 31: Flujo de datos del operador List Difference

3.2.3.5 Operador Calculate Tendency

Este operador permite realizar cálculos sobre los elementos de una lista, permitiendo calcular el máximo y mínimo, media aritmética, moda, mediana, desviación

típica, sumatorio y número de elementos de la lista recibida como entrada, enviando cada uno de estos valores por una salida distinta.

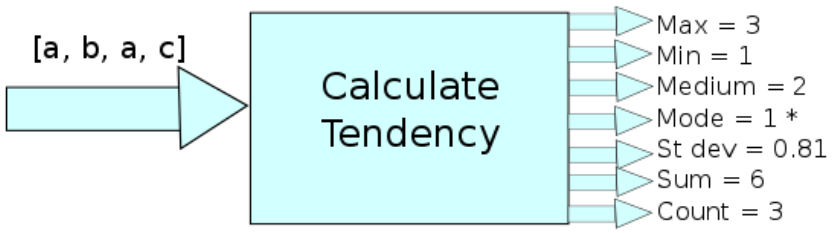


Figure 32: Flujo de datos del operador Calculate Tendency

3.2.3.6 Operador Labels to Dataserie

Este operador es el encargado de obtener una lista de valores asociada a una lista de etiquetas, de tal forma que estos valores obtenidos se corresponden con el número de apariciones de cada etiqueta.

Este operador espera como entrada una lista de etiquetas, y las salidas que genera a partir de ella son la lista descrita en el párrafo anterior junto a otra lista que contiene una única aparición de cada etiqueta, de forma que al estar en el mismo orden que las usadas para la lista de la otra salida, pueda usarse para identificar a qué etiqueta se corresponde cada valor de la primera salida.

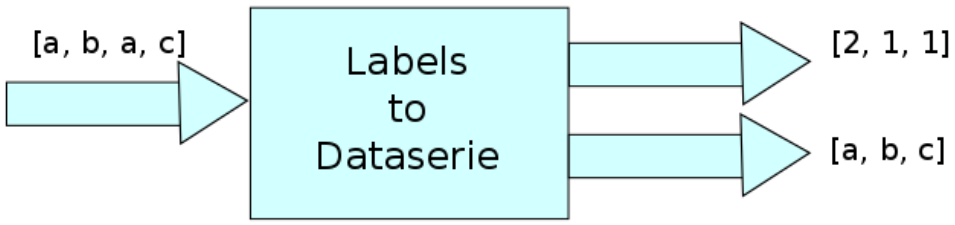


Figure 33: Flujo de datos del operador Labels to Dataserie

3.2.4 Splitters

Los splitters son operadores que generan a partir de una lista de objetos, un conjunto de listas de propiedades del objeto objetivo, permitiendo obtener dichas propiedades fácilmente y así poder ser usadas con el resto de componentes disponibles. Por tanto, las listas generadas por los operadores de tipo splitter tendrán siempre la misma longitud que la lista de objetos recibida y estarán ordenadas en el mismo orden que la entrada, de forma que se mantiene la correspondencia entre la cada elemento de la lista de entrada con los elementos en la misma posición de la lista en las salidas.

Estos operadores son muy importantes ya que permiten obtener tipos de datos más sencillos a partir de tipos complejos, como son los obtenidos por los harvesters. Por ejemplo, permite obtener, a partir de la lista de issues obtenida por los harvesters de issues, una lista de los usuarios asignados a dichas issues. Esto permite que los demás componentes de la aplicación de mashup puedan trabajar con datos más sencillos, de forma que se evita necesitar de un operador para cada resultado concreto, pudiendo, continuando con el ejemplo anterior, usar un operador que genere gráficas de tarta en función de una lista de palabras (tags) para obtener la gráfica de workload del proyecto, en vez de necesitar un operador concreto para esto.

3.2.4.1 Issue splitter

Este splitter es el encargado de obtener las propiedades de las issues normalizadas generadas por los harvesters de issues: Jira, Github y Gitlab harvesters.

Las salidas generadas por el issue splitter son: Una lista de estados de las issues, una lista de prioridades y de tipos de las issues (La prioridad y el tipo de issue son propiedades que sólo existen para las issues de Jira), el usuario asignado a cada issue, el sprint al que están asignadas y el mes en el que fueron creadas.

3.2.4.2 Operador Build splitter

Este splitter está dedicado a obtener las propiedades que tienen los objetos de tipo build obtenidos por el harvester de Jenkins.

Como salida genera las listas de ids, duraciones, cambios, causas de la build, información de los tests pasados y los timestamps asociados a las builds recibidas como entrada.

3.2.4.3 Operador Changes splitter

Este operador es el encargado de obtener las propiedades de los cambios realizados por una build de Jenkins. Esta entrada por tanto, es obtenida a partir del Build splitter definido previamente.

Las salidas que genera este splitter son las listas de ids del cambio, ids de commit, autores, comentarios, fechas de los cambios, y los ficheros afectados por los cambios.

3.2.4.4 Operador Test report splitter

Este operador obtiene los parámetros del informe sobre las build de Jenkins, que son obtenidos por el build splitter.

Las salidas de este operador son la lista de tests pasados, fallados y no realizados.

3.2.5 Transformación de datos para modelo visual

Estos operadores se encargan de preparar los datos para su representación visual, siendo el paso previo a esta. Estos generan las entradas esperadas por los componentes de representación visual.

3.2.5.1 Operadores para generar gráficas

Este tipo de operadores generan el modelo de datos esperado por el widget encargado de la representación de gráficas, el widget Highcharts, de forma que siempre se les conectará su salida a dicho operador.

3.2.5.1.1 Operador Column Chart Generator

Este operador está diseñado para generar gráficas de barras a partir de una lista de valores numéricos.

Las entradas que admite son la lista de valores numéricos a representar, y una lista de etiquetas con las que nombrar a cada una de las columnas generadas, este último siendo opcional usarlo.

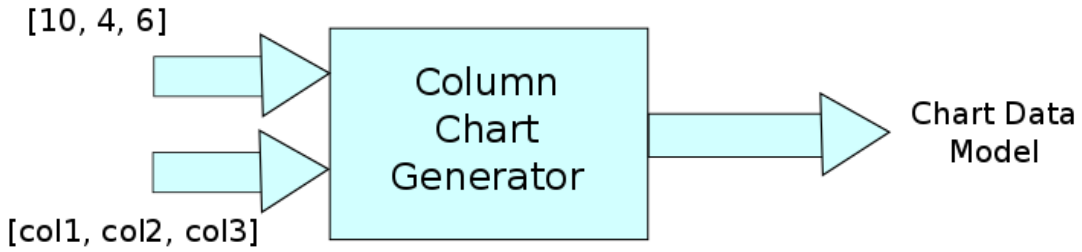


Figure 34: Flujo de datos del operador Column Chart Generator

3.2.5.1.2 Operador Pie Chart Generator

Este operador es el encargado de generar gráficas de tarta.

Tiene dos modos de funcionamiento en función de las entradas recibidas, pudiendo obtener una lista de valores, usando estos valores como tamaño de cada uno de los sectores de la gráfica, o recibiendo una lista de etiquetas que contará las apariciones de cada etiqueta para calcular el tamaño de los sectores asociados a la etiqueta.

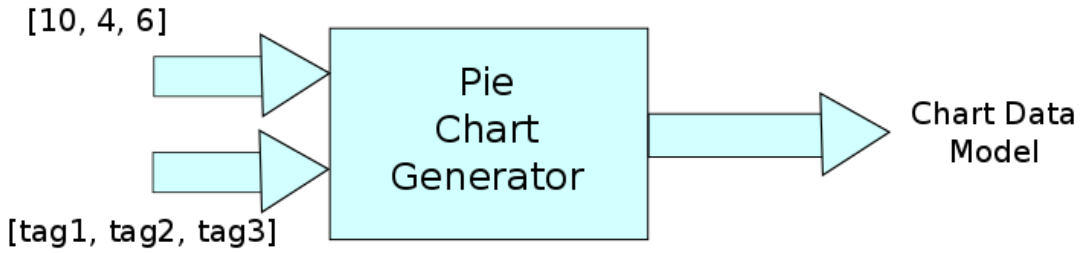


Figure 35: Flujo de datos del operador Pie Chart Generator

3.2.5.1.3 Operador Trend Chart Generator

Este operador es el encargado de generar gráficas de líneas, con cualquier número de series de valores en ella.

La entrada principal de este operador es la lista de valores asociados a la serie de valores mostrada, esta entrada puede ser empaquetada previamente usando el operador packlist definido previamente, de forma que reciba varias series de valores en la misma entrada y las represente todas en el gráfico resultante.

También puede recibir una lista de timestamps o etiquetas con las que nombrar los puntos del eje X.

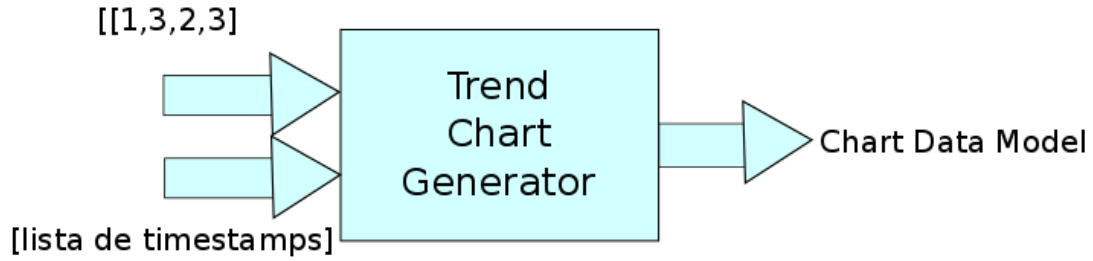


Figure 36: Flujo de datos del operador Trend Chart Generator

3.2.5.1.4 Operador Burndown Chart Generator

Operador encargado de generar la gráfica burndown de un proyecto ágil asociada a un sprint del proyecto.

La entrada esperada por este operador es la lista de issues de uno de los sprints. Esto se puede conseguir fácilmente filtrando la salida de los harvesters de issues usando el operador and filter y el widget set generic filter conditions.

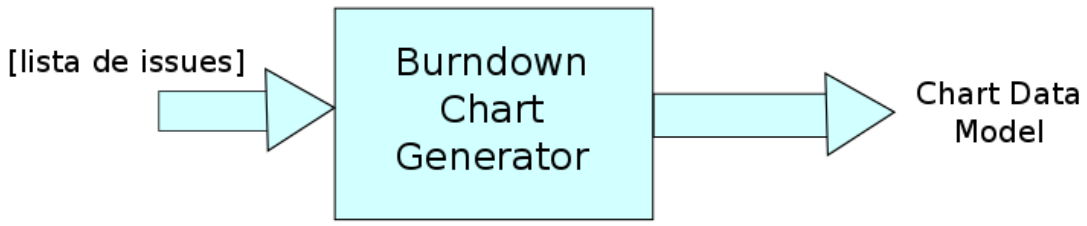


Figure 37: Flujo de datos del operador Burndown Chart Generator

3.2.5.1.5 Operador Workload Chart Generator

Este operador es el encargado de generar la gráfica de workload asociada a un conjunto de issues o commits, estos pudiendo ser todos los issues/commits del proyecto, o solo una parte de ellos, como por ejemplo los issues de un sprint. Tiene dos entradas posibles, una para issues en caso de que se quiera generar el workload a partir de issues, y otro para commits para generar el workload a partir de los commits.

Si todos los elementos elementos de entrada pertenecen al mismo mes, el resultado será un gráfico de tarta, habiendo una sección por cada miembro del proyecto que tiene asignado o es autor de los elementos. En caso contrario, si hay elementos de asociados a varios meses, se generará un gráfico de columnas, con una columna por cada miembro y mes.

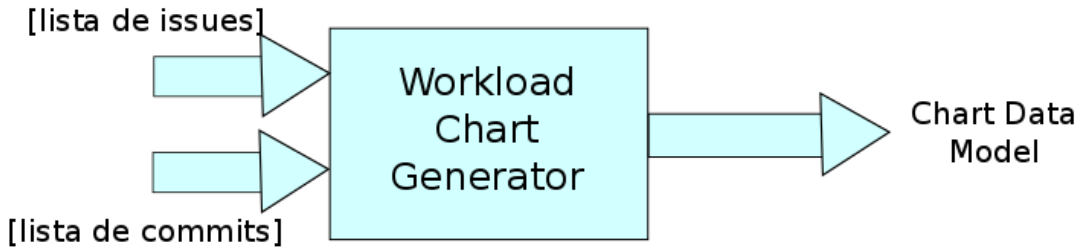


Figure 38: Flujo de datos del operador Workload Chart Generator

3.2.5.1.6 Operador Product Backlog Evolution Chart Generator

Este operador genera la gráfica del backlog evolution del proyecto analizado. Por tanto, espera recibir como entrada la lista de issues asociada al proyecto analizado, aunque también se puede usar enviando un subconjunto de issues para obtener solo la información relativa a ellos.

3.2.5.1.7 Operador Reliability Chart Generator

Este operador se encarga de obtener la gráfica de reliability de un proyecto, permitiendo mostrar la relación entre issues abiertos y cerrados asignados a un miembro del proyecto.

Dependiendo de la entrada recibida, la salida puede ser un gráfico de barras, si existen issues relacionados con más de un miembro del proyecto, en cuyo caso el resultado mostrará una columna por cada miembro del proyecto, siendo estas columnas compuestas por dos partes: las issues abiertas y las issues cerradas. En el caso de que solo haya issues asignadas a un único miembro del proyecto, el resultado será una gráfica de tarta con dos únicos sectores, siendo estos las issues abiertas y las cerradas.

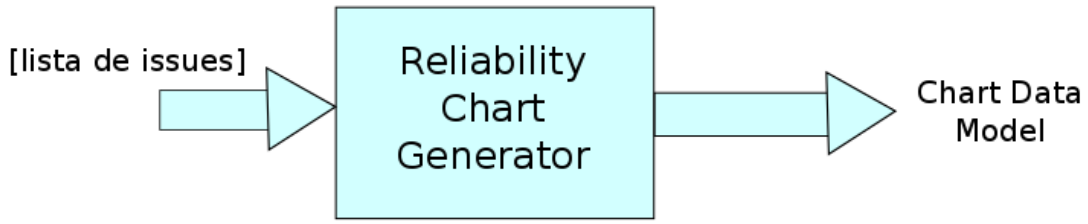


Figure 39: Flujo de datos del operador Reliability Chart Generator

3.2.5.1.8 Operadores para generar tablas

Este tipo de operadores generan el modelo de datos que espera recibir el widget data-viewer, encargado de generar tablas de datos, por lo que su salida siempre estará conectada a un widget de este tipo.

3.2.5.1.9 Operador Issue table generator

Operador encargado de generar el modelo de datos necesario para representar los issues recibidos en una tabla de datos. Las columnas de las que dispondrá la tabla generada serán los siguientes: el id de la issue, el título de la issue, el estado de la issue, el miembro del proyecto asignado a al issue, el sprint al que pertenece la issue y las fechas de creación y resolución y la fecha límite para cerrar la issue.

Este operador espera recibir como entrada una lista con cualquier número de issues de cualquiera de los harvesters de issues definidos previamente.

3.2.5.1.10 Operador Jenkins project build list

Este operador es el encargado de generar las tablas con las builds obtenidas por el harvester de Jenkins. La tabla resultante contendrá las siguientes columnas: Resultado de la build (si la build fue correcta, falló o fue abortada), el id de la build y la fecha de la build.

La entrada de este operador será la lista de builds obtenida por el harvester de Jenkins.

3.2.6 Representación visual

3.2.6.1 Widget Highcharts

Este widget es el encargado de la representación gráfica de tablas. Utiliza Highcharts, una librería de representación de gráficas para Javascript.

Este widget espera como entrada el set de datos que define la gráfica a generar, ocupándose de configurar la librería para su correcta representación. Este set de datos es generado por operadores como el Burndown chart generator.

Las gráficas generadas por este operador permiten seleccionar puntos de las mismas, que serán enviados por su salida al wiring, de forma que puedan ser aprovechadas por otros componentes

3.2.6.2 Widget Dataviewer

Este es el widget encargado de la representación de tablas de datos, actuando de interfaz sobre las tablas proporcionadas de la plataforma Wirecloud.

Este Widget espera una entrada, siendo esta los datos y la estructura de la tabla a mostrar. Esta entrada es generada por operadores como el Issue table generator y contiene la estructura que tendrá la tabla, es decir, las columnas y los tipos de datos a representar, así como los valores que han de ser representados en la tabla generada.

Estas tablas permiten filtrar los datos en función a palabras clave, ya sea por columnas o en toda la tabla, a la vez que permite ordenar los datos en orden creciente o decreciente en base a la columna seleccionada.

También es posible seleccionar filas de la tabla, de forma que el widget enviará por sus salidas los elementos seleccionados y las condiciones de filtro que requiere el operador And Filter.

3.2.6.3 Widget Panel

Este widget es el encargado de mostrar en la vista del usuario un valor, ya sea un número o un texto. Esto puede ser útil para mostrar valores concretos, como el número de issues con etiqueta bug que hay en todo el proyecto.

Este widget recibe como entrada el valor que tiene que representar, tras lo cual lo representa el valor recibido en la vista del usuario.

3.2.7 Interacción con el usuario

3.2.7.1 Operador Open website

Este operador permite abrir páginas web como una pestaña a partir de la información usada en la entrada. Esto es especialmente útil cuando se usa en conjunto con una gráfica o una tabla para abrir la web de una issue y así poder hacer modificaciones sobre la misma.

La entrada de datos esperada por este operador es una lista de elementos o un elemento cualquiera que tenga la propiedad "link". El valor de esa propiedad será la uri que se usará en las pestañas a abrir.

3.2.7.2 Operador Burndown click

Este operador es el encargado de obtener, a partir del día seleccionado por el usuario en la gráfica de burndown, las issues que fueron cerradas dicho día.

Este operador espera como entrada el mismo conjunto de issues que recibió el operador burndown chart generator y la información que envía sobre el click del usuario el widget highcharts.

3.3 Casos de uso

Usando los componentes que se han desarrollado, es posible generar una gran variedad de tablas y gráficas sobre la información de un proyecto ágil. En este apartado se documentarán algunos ejemplos de uso, desde los más básicos como el filtrado de los datos disponibles, a cosas más complejas como obtener los responsables de líneas de código o mezclar información de varias herramientas ágiles.

3.3.1 Filtrado de datos

El filtrado de datos diseñado está basado en el uso del operador And Filter, cuyo uso estará casi siempre acompañado por el widget Generic Filter Conditions siguiendo la estructura mostrada en la figura 40.

Una vez configurado los componentes en el wiring, en la vista del workspace se podrán elegir los filtros a aplicar a los datos recibidos usando los desplegables asociados a cada tipo de filtro, siendo su valor por defecto "all" que haría que ese filtro no se aplique al aceptar cualquier elemento recibido. La figura 41 muestra la vista de workspace del ejemplo creado en la figura 40, habiendo sido seleccionado un filtro para el sprint de mayo.

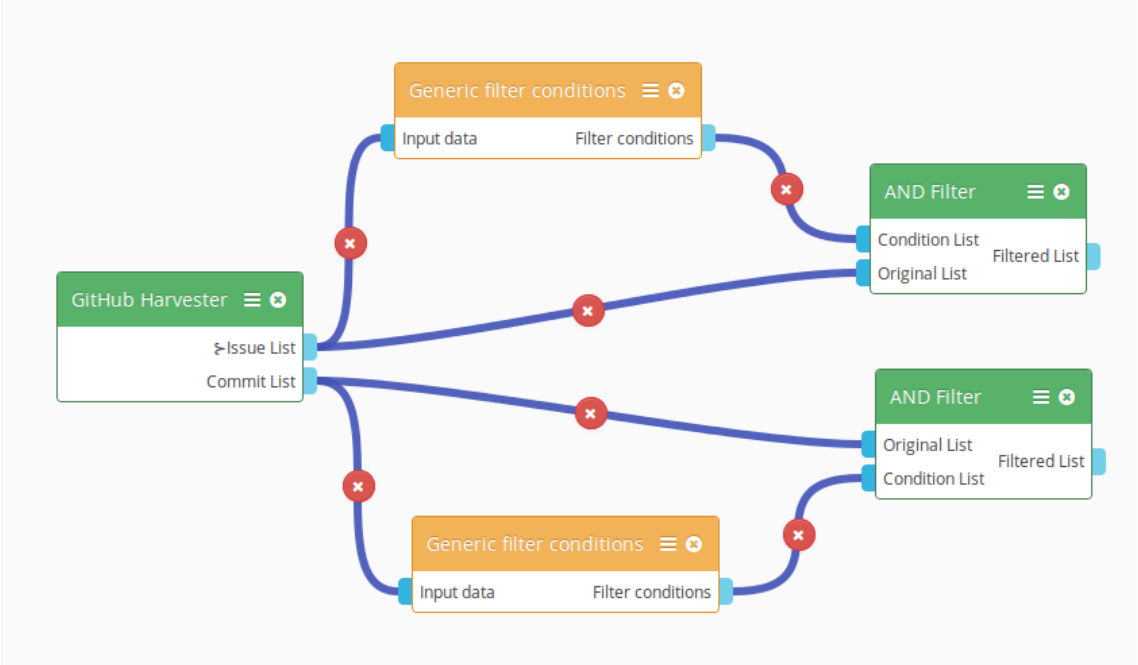


Figure 40: Vista de Wiring de ejemplo de filtrado de datos

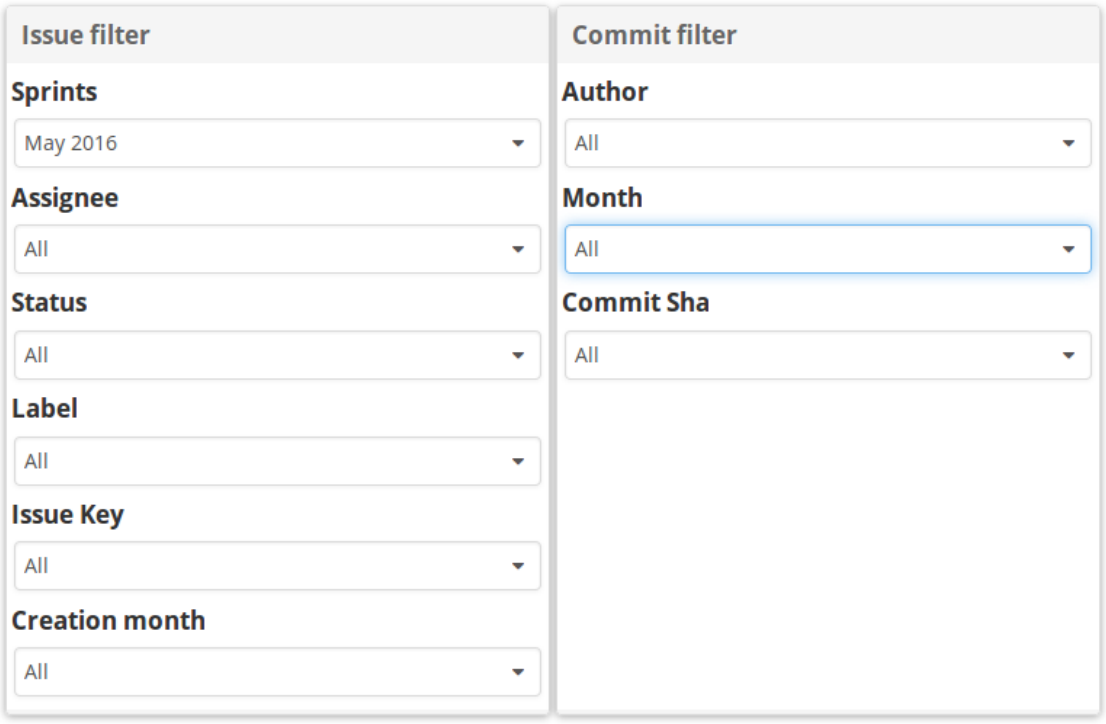


Figure 41: Vista de workspace del ejemplo de filtrado de datos

3.4 Tendencia de defectos en el proyecto

La tendencia de defectos representa el número de defectos encontrados en el proyecto a lo largo del tiempo, en este caso representada en una de tabla de columnas.

Esto se puede conseguir fácilmente, añadiendo al harvester de issues elegido un filtro que seleccione solo las issues que sean de tipo "Bug" (error) y tras esto simplemente mostrar en la gráfica el número de issues de cada mes, esta última parte se puede conseguir con el uso del splitter de issues conectándolo al operador Label to Dataserie de forma que convierta una lista de nombres de meses en una lista que cuenta las apariciones de cada mes, representado luego esta lista usando el operador Column Chart Generator. En la figura 42 se puede observar la configuración del wiring de los componentes que ha sido descrita, y en la figura 43 se puede observar cómo se sería el resultado en la vista del workspace.

El mashup desarrollado puede ser usado para contar el número de issues por mes que cumplan cualquier condición, simplemente cambiando la configuración del filtro, permitiendo mostrar otras informaciones como por ejemplo la cantidad de issues que

ha habido en cada mes o la carga de trabajo de un único trabajador a lo largo de los meses

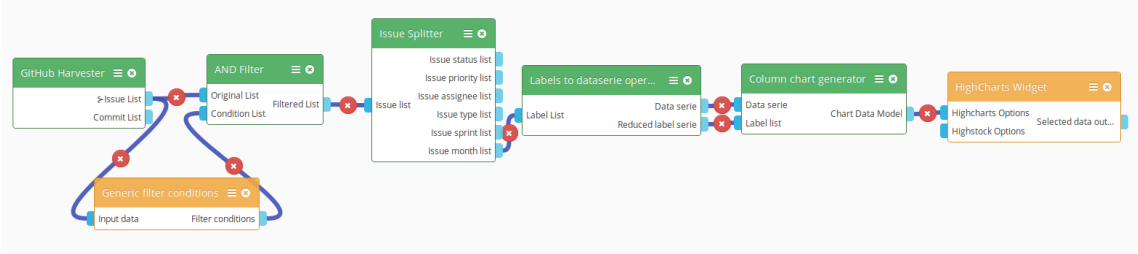


Figure 42: Vista de Wiring del ejemplo de tendencia de defectos

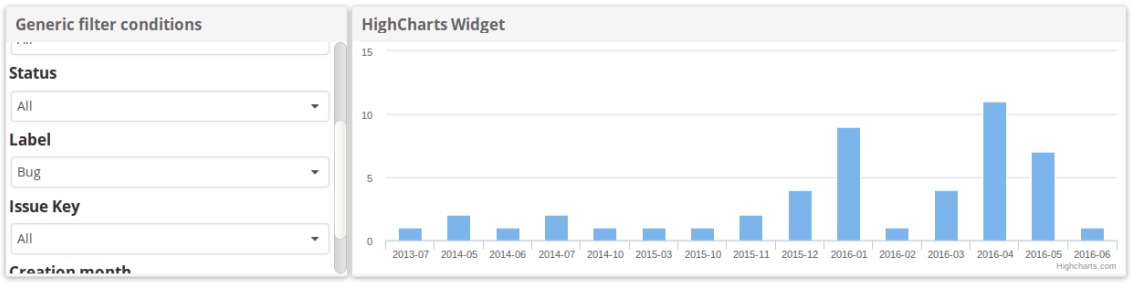


Figure 43: Vista de workspace del ejemplo de tendencia de defectos

3.4.1 Tendencia de build fallidas de Jenkins

Para obtener la tendencia de build pasadas y fallidas se usará el harvester de Jenkins junto al splitter de builds, que permitirá obtener lalista de tests, de la cual, usando el splitter de test reports, se podrá obtener la lista de builds pasadas, falladas e ignoradas. Con estos datos, junto al operador Trend Chart Generator se puede mostrar una gráfica con la tendencia de resultados de las builds. En este caso se desea mostrar tanto las pasadas como las falladas, por lo que se agruparán usando el operador Packlist como se muestra en la figura 44.

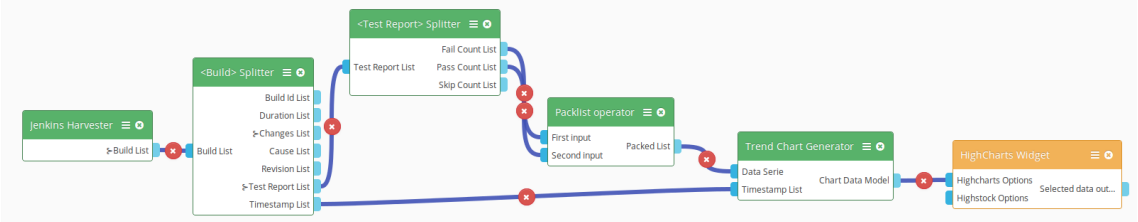


Figure 44: Vista de Wiring del ejemplo de la tendencia de builds fallidas

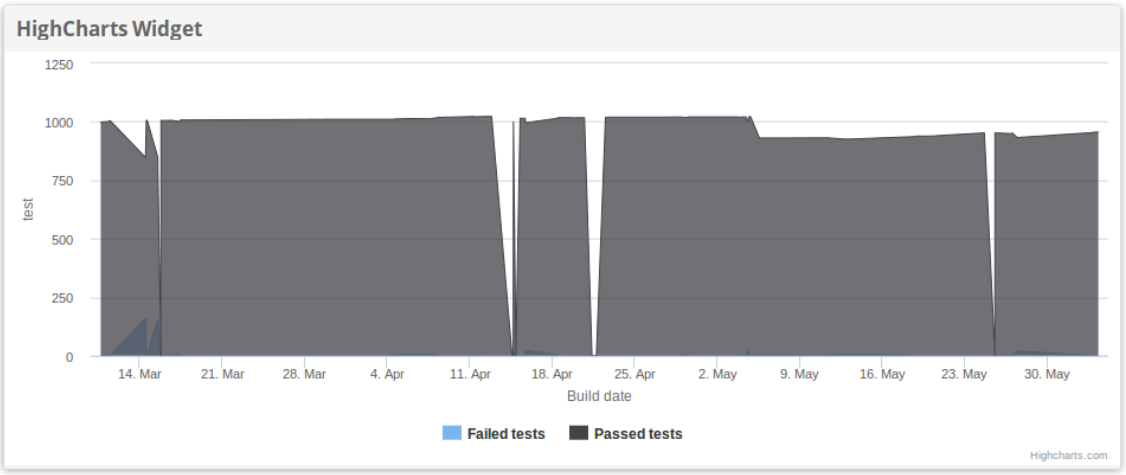


Figure 45: Vista de workspace del ejemplo de la tendencia de builds fallidas

3.4.2 Gráfica Burndown de un sprint

Para poder obtener la gráfica burndown de un proyecto, es necesario haber añadido y configurado el harvester de la herramienta ágil usada en el proyecto y un filtro de datos como el descrito en el apartado 3.3.2.

Para generar la gráfica burndown se usará el operador Burndown Chart Generator, conectando su entrada a la salida del operador And Filter usado para seleccionar el sprint deseado de entre los issues obtenidos por el harvester, y la salida al widget Highcharts, que será el encargado de representarla en la vista del usuario. Con esto ya se genera y muestra la gráfica burndown.

En caso de que se quiera obtener las issues de la gráfica sobre las que el usuario ha hecho click y representarlas en una tabla, bastaría con añadir los operador Burndown Chart Click y Issue Table Generator y el widget Data Viewer y conectarlos como se muestra en la figura 46 que documenta el anterior ejemplo.

Para usar el mashup de este ejemplo, simplemente hay que seleccionar un sprint en el filtro añadido, lo cual crearía la gráfica burndown asociada a dicho sprint. Si se selecciona dentro de la gráfica uno de los días, las issues que fuesen cerradas el día seleccionado serán mostradas en la tabla. En la figura 47 se puede observar este ejemplo habiendo seleccionado en el filtro el sprint de abril y habiendo seleccionado en la gráfica burndown el día 8.

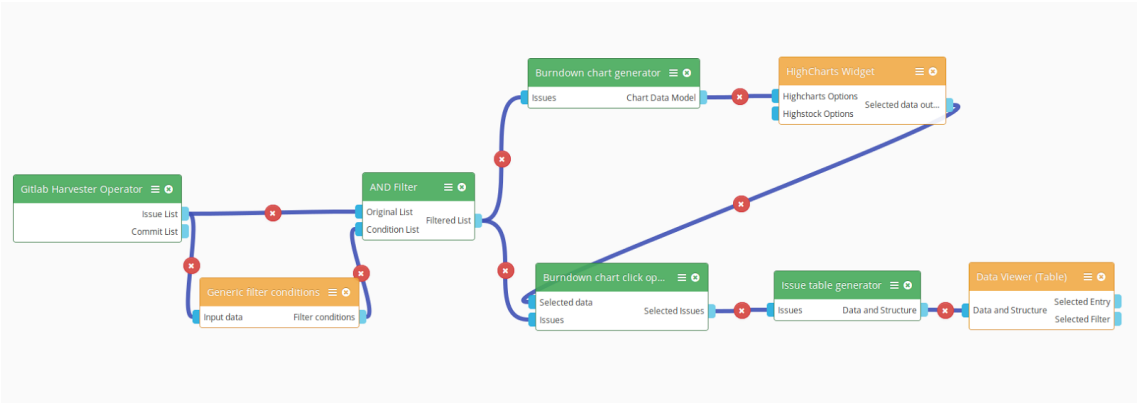


Figure 46: Vista de Wiring del ejemplo de la gráfica Burndown

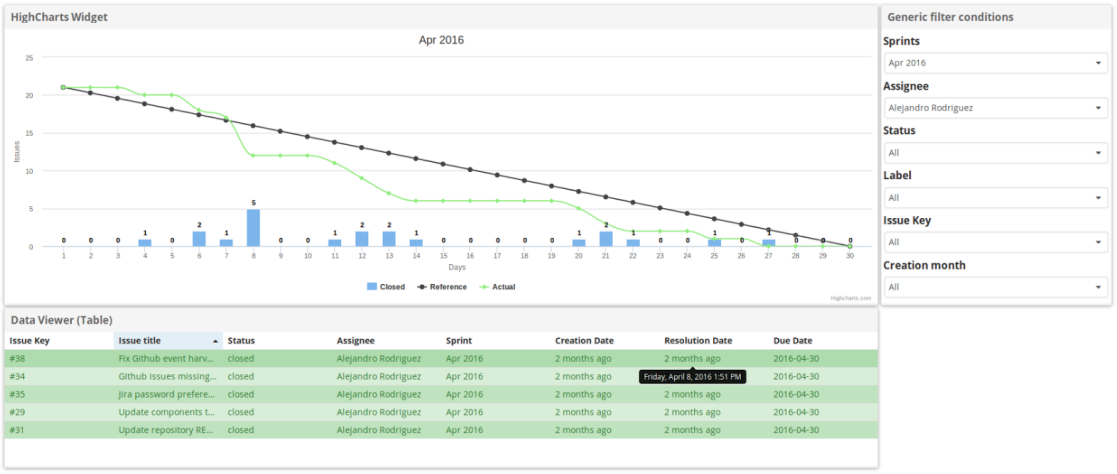


Figure 47: Vista de workspace del ejemplo de la gráfica Burndown

3.5 Git blame de una issue

En el caso de que se resuelva un error mediante una issue, puede ser útil saber qué líneas de código causaban el error y quien era el autor de dichas líneas erróneas. En este ejemplo se muestra como obtener dicha información, usando el harvester de Github para obtener las issues que posteriormente serán filtradas, usando la estructura de filtrado descrita anteriormente, para obtener solo las que son de tipo “Bug”, tras esto, al ser mostradas en una tabla gracias al operador Issue Table Generator, permiten elegir de entre las disponibles la issue a analizar, obteniendo de ella el commit que la cerró usando el operador Get Closing Commit y enviándolo al

operador Git Blame, que obtendrá las líneas modificadas y preparará la estructura de datos esperada por el Data Viewer para mostrar esta información en una tabla.

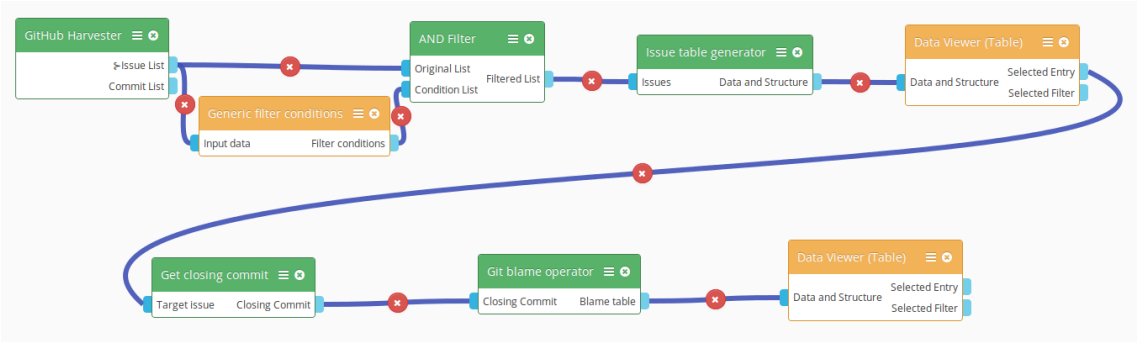


Figure 48: Vista de Wiring de ejemplo del Blame de una issue

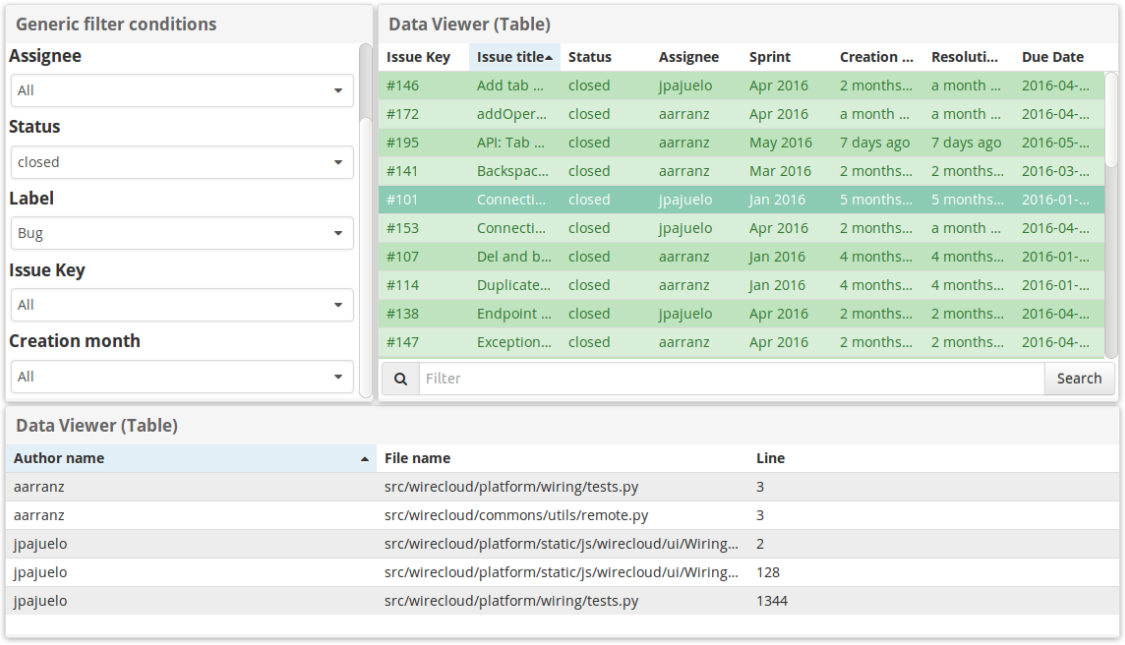


Figure 49: Vista de workspace de ejemplo del Blame de una issue

3.6 Diferencia en la duración de los tests pasados entre builds de jenkins

Para poder comparar la duración de los tests pasados por dos builds diferentes se usará el harvester de Jenkins junto a algún método para seleccionar únicamente

dos builds. En este caso, se usarán dos tablas para seleccionar las builds. Una vez seleccionadas se creará una lista con ellas usando el operador Packlist, que posteriormente será enviada al operador Detailed Jenkins Test Report, que obtendrá más datos sobre los tests pasados por estas builds. Una vez obtenida esta información, se enviará al operador Jenkins Test Time Difference, que al enviar su salida al widget Data Viewer, creará una tabla con información sobre la duración de cada uno de los tests pasados por las builds seleccionadas, permitiendo observar la diferencia de tiempo entre ellas.

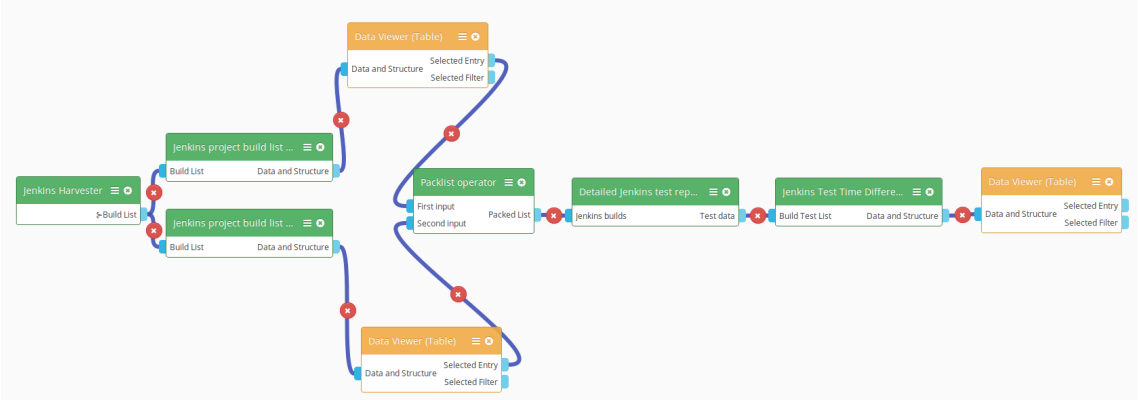
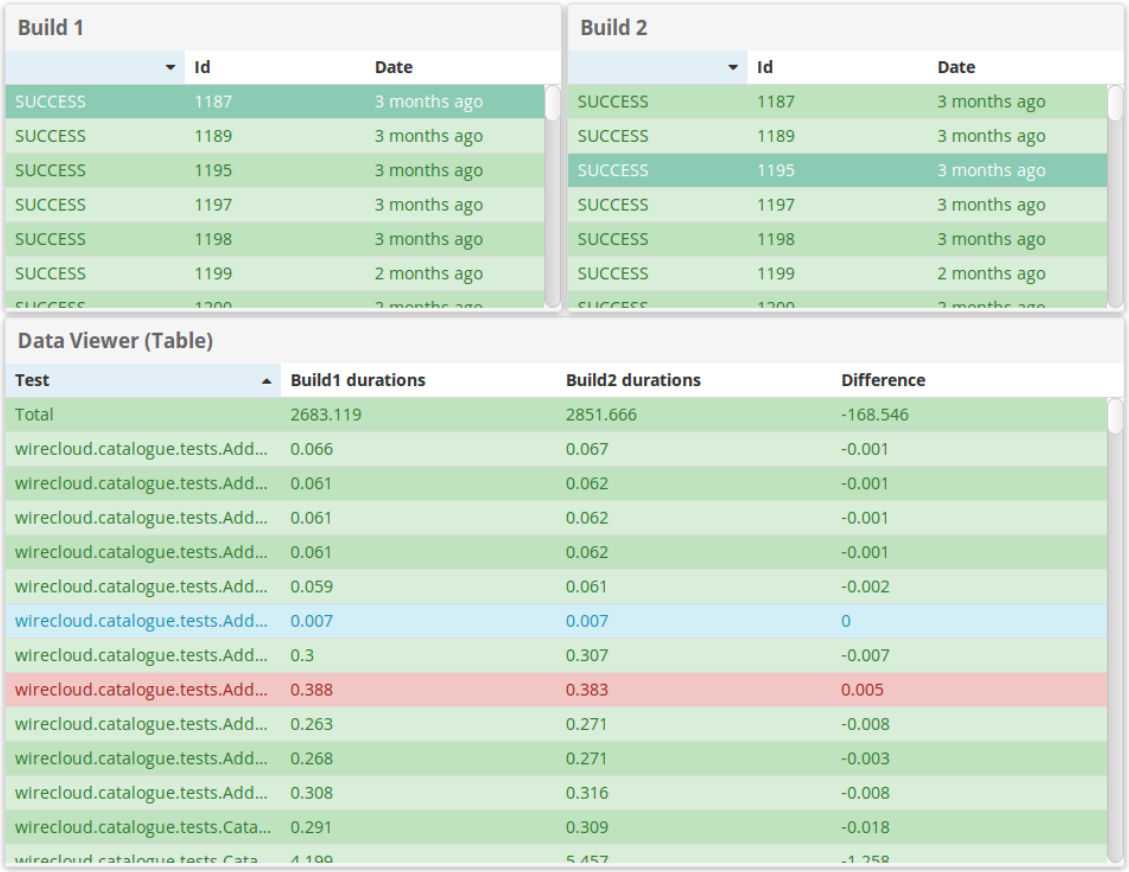


Figure 50: Vista de Wiring del ejemplo de diferencia del tiempo de tests



Build 1

	Id	Date
SUCCESS	1187	3 months ago
SUCCESS	1189	3 months ago
SUCCESS	1195	3 months ago
SUCCESS	1197	3 months ago
SUCCESS	1198	3 months ago
SUCCESS	1199	2 months ago

Build 2

	Id	Date
SUCCESS	1187	3 months ago
SUCCESS	1189	3 months ago
SUCCESS	1195	3 months ago
SUCCESS	1197	3 months ago
SUCCESS	1198	3 months ago
SUCCESS	1199	2 months ago

Data Viewer (Table)

Test	Build1 durations	Build2 durations	Difference
Total	2683.119	2851.666	-168.546
wirecloud.catalogue.tests.Add...	0.066	0.067	-0.001
wirecloud.catalogue.tests.Add...	0.061	0.062	-0.001
wirecloud.catalogue.tests.Add...	0.061	0.062	-0.001
wirecloud.catalogue.tests.Add...	0.061	0.062	-0.001
wirecloud.catalogue.tests.Add...	0.059	0.061	-0.002
wirecloud.catalogue.tests.Add...	0.007	0.007	0
wirecloud.catalogue.tests.Add...	0.3	0.307	-0.007
wirecloud.catalogue.tests.Add...	0.388	0.383	0.005
wirecloud.catalogue.tests.Add...	0.263	0.271	-0.008
wirecloud.catalogue.tests.Add...	0.268	0.271	-0.003
wirecloud.catalogue.tests.Add...	0.308	0.316	-0.008
wirecloud.catalogue.tests.Cata...	0.291	0.309	-0.018

Figure 51: Vista de workspace del ejemplo de diferencia del tiempo de tests

3.7 Worload y backlog evolution de un proyecto

En este ejemplo se muestra la flexibilidad ofrecida por los componentes que se han desarrollado, permitiendo obtener el workload de un proyecto en una gráfica de columnas y tartas. También se representará la gráfica de backlog evolution del proyecto, que permite observar la evolución del mismo en cuanto a cantidad de issues.

Para este ejemplo se usará el harvester de Github, generando el workload de columnas usando el operador Workload Chart generator (el resultado de este será un gráfico de columnas al haber más de un mes en los issues recibidos como entrada). Para generar el workload en un gráfico de tartas se usará el splitter de issues, enviando la lista de assignee (usuarios a los que están asignadas las issues) al operador Pie Chart generator, resultando en una gráfica de workload en la que además se mostrará el porcentaje de issues sin asignar. Por último, para generar la gráfica del

backlog evolution de todo el proyecto, simplemente se enviará la salida del harvester de Github al operador responsable de generar la misma: el Product Backlog Chart.

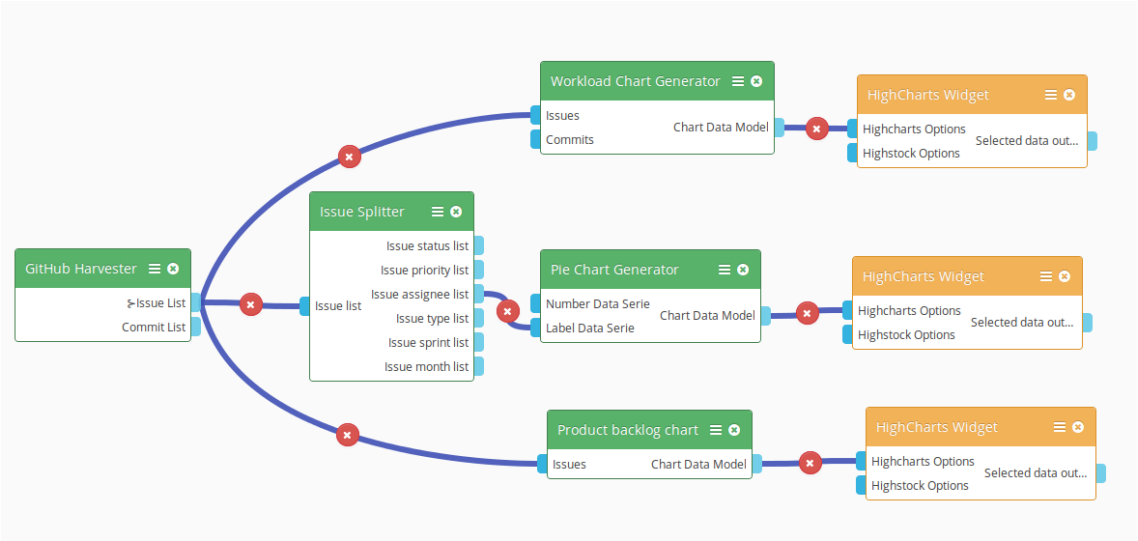


Figure 52: Vista de Wiring de ejemplo de workload y backlog evolution

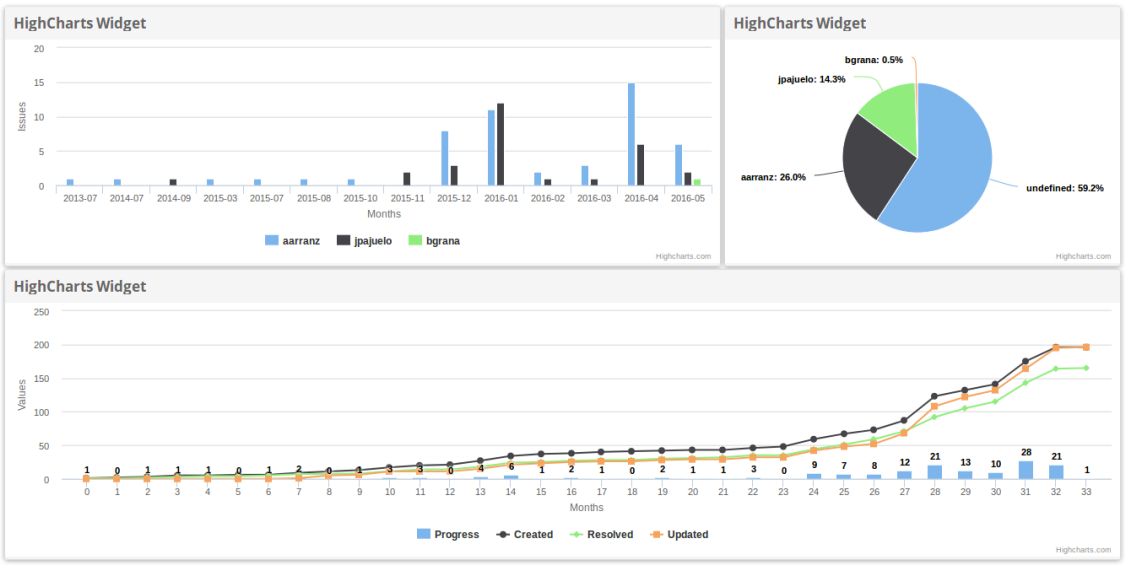


Figure 53: Vista de workspace de ejemplo de workload y backlog evolution

3.8 Mashup de datos obtenidos de varias herramientas ágiles

En este ejemplo se muestra una de las funciones principales de una aplicación mashup: juntar datos de varias fuentes en uno mismo para producir resultados a partir de dicho conjunto, con esto en mente, los harvesters de Issues que se han desarrollado permiten mezclar sus salidas y usarlas como una sola. Esto se puede realizar usando operador como el operador Packlist o el operador Union List. En este caso se usarán el harvester de Github junto al harvester de Jira, uniéndolos usando el operador Union List. Tras esto, la información combinada de ambos harvesters se envía a un issue splitter, del cual se obtiene la lista de assignees para enviársela al operador Pie Chart, que generará una gráfica de workload. Esta lista de issues combinada también se enviará al operador Reliability Chart Generator, obteniendo así la gráfica de fiabilidad de los proyectos.

Se puede apreciar que la gráfica de workload resultante en este ejemplo es distinta de la mostrada en el ejemplo anterior. Esto es porque esta incluye también los issues almacenados en la herramienta ágil Jira.

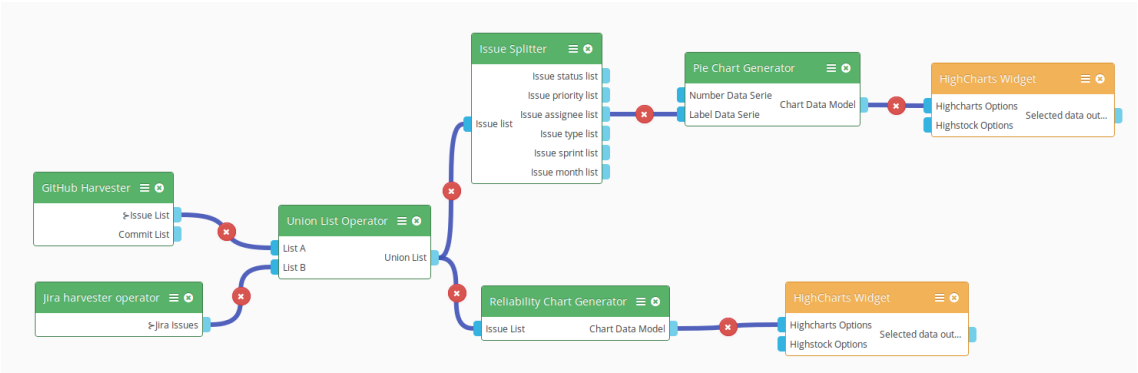


Figure 54: Vista de Wiring de ejemplo de mashup de issues de Github y Jira

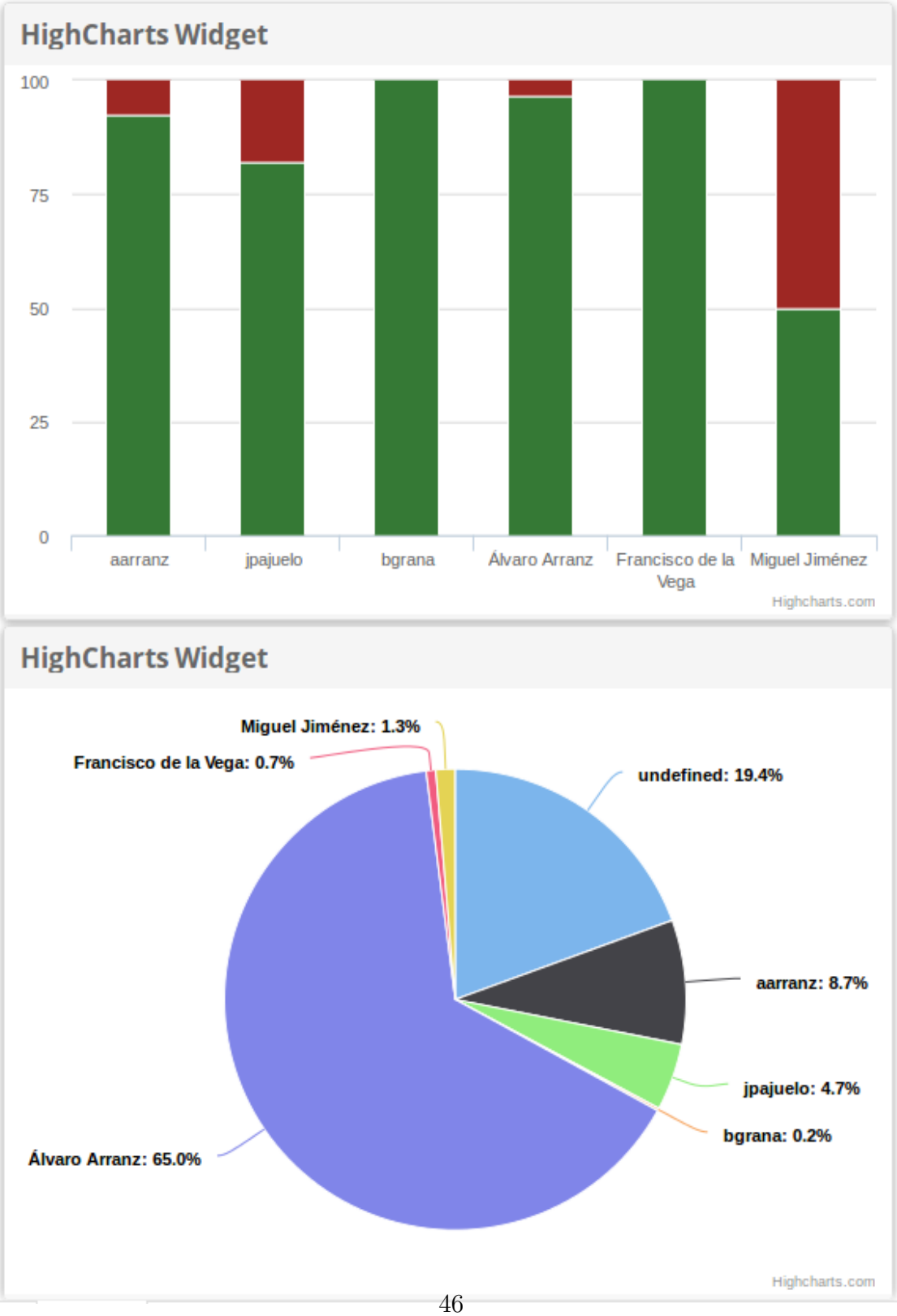


Figure 55: Vista de workspace de ejemplo de mashup de issues de GIthub y Jira

4 CONCLUSIONES Y LINEAS FUTURAS

En este apartado se planetan las conclusiones obtenidas al haber terminado el proyecto y posibles líneas futuras sobre elementos que podrían implementarse para mejorar la calidad final de los mashups y componentes desarrollados.

4.1 Conclusiones

Los componentes desarrollados permiten representar toda la información que había disponible en la aplicación que se usaba en el caso de uso planteado, dando a su vez libertad para añadir información de otras herramientas ágiles y permitiendo modificar fácilmente la configuración del mashup para generar información que pueda ser de valor para el usuario. Los componentes desarrollados son en su mayor parte genéricos, permitiendo su uso fuera de los casos de uso planteados, dando así mayor libertad al usuario final.

Los componentes desarrollados siguen una orientación hacia un modelo de vista-controlador, estando los mashups desarrollados muy centrados al uso de operadores y su configuración dentro de la vista de Wiring, estando muy dividida la interfaz de usuario de la lógica de la aplicación, al no realizar ninguna operación los widgets y delegar todas las operaciones a los operadores. Sin embargo, el uso tradicional de Wirecloud está orientado a la interacción entre los widgets, habiendo muchos menos operadores y realizando gran parte de las operaciones en los widgets.

4.2 Líneas futuras

En cuanto a líneas futuras de trabajo, se plantea añadir más harvesters de datos, dando así soporte a una mayor cantidad de herramientas ágiles, ya que aunque se da soporte a las más usadas, quedan muchas más por soportar, de forma que se limite menos las posibilidades del usuario.

También se plantea el diseño e implementación de un operador Splitter que sea genérico, de forma que usando únicamente este nuevo splitter, se pueda extraer información de los datos recibidos, independientemente de su origen, aunque esto ahora mismo no es posible con las características de la plataforma Wirecloud al no ser posible crear de forma dinámica conexiones de wiring.

References

[1] LOELIGER, Jon, *Version Control with Git: Powerful tools and techniques for collaborative software development*, 2012

[2] Schwaber, Ken, *Agile project management with Scrum*, 2004

[3] Página web de eXtreme Programming: http://www.extremeprogramming.org/ accedida por última vez en junio de 2015

[4] Página web de Highcharts: http://www.highcharts.com/ accedida por última vez en junio de 2015

[5] Página web de Git: https://git-scm.com/ accedida por última vez en junio de 2015

[6] Página web de FIWARE https://www.fiware.org/ accedida por última vez en junio de 2015

[7] Página web del laboratorio CoNWeT http://conwet.fi.upm.es/ accedida por última vez en mayo de 2015

[8] Página web de Wirecloud: http://conwet.fi.upm.es/wirecloud/ accedida por última vez en mayo de 2015

[9] Descripción de Wirecloud en el catálogo de FIWARE: http://catalogue.fiware.org/enablers/application-mashup-wirecloud accedida por última vez en junio de 2015

[10] Página web de Github: https://github.com/ accedida por última vez en junio de 2015

[11] Página web de Gitlab: https://about.gitlab.com/ accedida por última vez en mayo de 2015

[12] Página web de Jenkins: https://jenkins.io/ accedida por última vez en junio de 2015

[13] Página web de Jira: https://es.atlassian.com/software/jira accedida por última vez en junio de 2015

[14] Página web de JQuery https://jquery.com/ accedida por última vez en junio de 2015

[15] Información sobre JavaScript https://en.wikipedia.org/wiki/JavaScript accedida por última vez en mayo de 2015

[16] Información sobre HTML https://es.wikipedia.org/wiki/HTML accedida por última vez en junio de 2015

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Mon Jun 06 18:20:43 CEST 2016
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)